Proyecto Fin de Grado
Grado en Ingeniería Aeroespacial
Intensificación de Navegación Aérea

Implementación optimizada de algoritmos de búsqueda en hardware avanzado para aplicaciones de alta potencia

Autor: Francisco Javier González Rodríguez

Tutor: Marta Laguna García

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Grado

Grado en Ingeniería Aeroespacial
Intensificación de Navegación Aérea

Implementación optimizada de algoritmos de búsqueda en hardware avanzado para aplicaciones de alta potencia

Autor:
Francisco Javier González Rodríguez

Tutor:
Marta Laguna García
Profesor Contratado

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Grado: Implementación optimizada de algoritmos de búsqueda en hardware avanzado para aplicaciones de alta potencia

Autor: Francisco Javier González Rodríguez
Tutor: Marta Laguna García

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar, me gustaría dedicar un afectuoso agradecimiento a mi tutora, Marta Laguna, así como a José Ignación León, por toda la ayuda y confianza que han demostrado en mi entrada al mundo de la Electrónica de Potencia. Nunca encontraré palabras suficientes para expresar mi gratitud de dejarme formar parte de este grupo de trabajo.

De igual forma, agradecer la labor de Ramón Portillo, por su invaluable labor técnica en preparar todo los equipamientos necesarios, así como a Leopoldo García Franquelo, Sergio Vázquez y Abraham Marquez por sus acertadas propuestas y sugerencias en la elaboración de este trabajo, con especial afecto a este último por su inspiradora labor crítica y de revisión de la documentación.

Agradecer por supuesto a mis padres, Tere y Paco, cuantísimos esfuerzos han tenido que asumir para que hoy esté aquí y sea quien soy. Ellos siempre sabrán que han sido mis modelos a seguir.

Me disculpor por no poder nombrar a todas aquellas personas que, de una forma u otra, han aportado su grano de arena a que este documento haya sido editado y no han sido recogidas aquí. No obstante, he de nombrar: a Joanna Pakuła -*she knows why!*-, a Marcelo Meneses Marín, por su implacable interés en todo y su gran capacidad para aportar la idea necesaria en el momento adecuado, a Fco. Garrocho López, por tantísimos momentos especiales en estos cuatro años de camino -*¡y los que quedan!*- y a Antonio Martín Morilla, por aquellas ideas que han surgido juntos al lado de una taza de café.

Por último, agradecer al Ministero de Educación, Cultura y Deporte y a la Universidad de Sevilla por su ayuda bajo el programa de 'Becas de Colaboración de estudiantes en departamentos universitarios para el curso académico 2016-2017'.

Resumen

El aumento incesante de la demanda de energía ha conllevado la instalación de sistemas de producción de energía de mayor potencia. Sin embargo, estos sistemas de alta potencia tienen unos parámetros muy estrictos en cuanto a eficiencia, robustez, tolerancia a fallos, fácil mantenimiento, etcétera.

En algunas aplicaciones de sistemas de alta potencia, una de las técnicas de modulación y control aplicadas es la eliminación selectiva de armónicos. Este método de control y modulación se basa en el estudio a priori del contenido armónico de las formas de onda de salida del convertidor, facilitando el diseño del filtro de salida del convertidor, pero limitando las pérdidas por conmutación del equipo, lo que hace que el rendimiento del mismo se mantenga por encima de los límites requeridos.

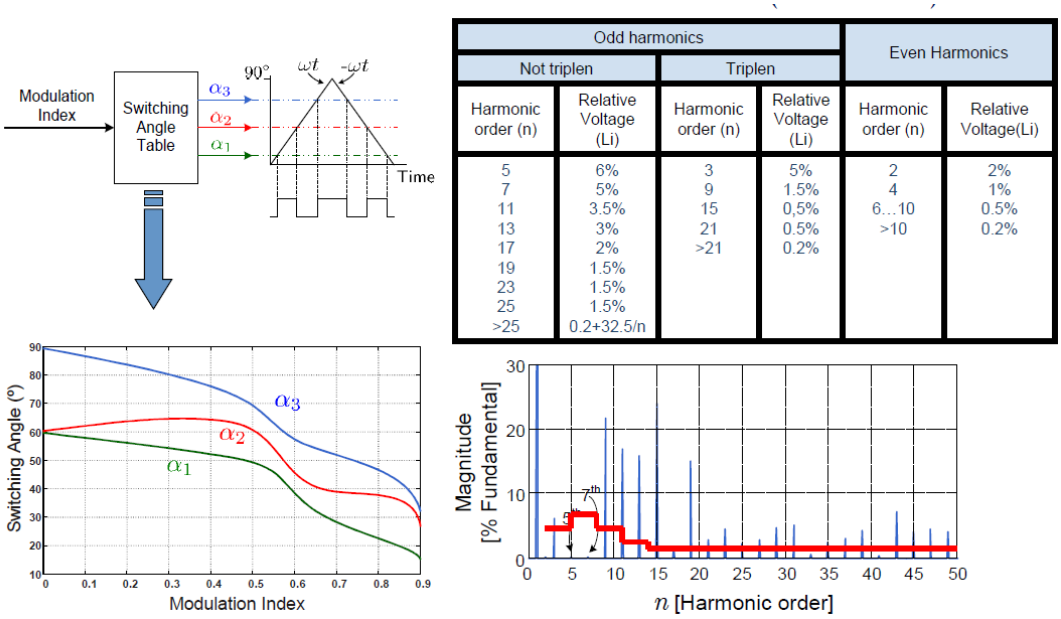
Las técnicas de eliminación selectiva de armónicos necesitan de una capacidad de cálculo enorme para llevar a cabo una gran cantidad de operaciones no lineales con el objeto de resolver un sistema de ecuaciones complejo. Existen en la bibliografía múltiples métodos de búsqueda matemáticos para resolver este tipo de sistemas de ecuaciones (se muestra un ejemplo en Fig. 1), pero la velocidad de resolución del mismo hace que necesariamente el proceso tenga que realizarse offline, no pudiéndose implementar en tiempo real, que sería lo más eficiente.

Además, recientemente se ha presentado la técnica de mitigación selectiva de armónicos, que plantea resolver un sistema de inecuaciones en vez de un sistema de ecuaciones. Esto hace que la búsqueda de soluciones sea aún más compleja.

La evolución de los sistemas microprocesadores en las últimas décadas ha hecho que se puedan realizar millones de instrucciones por segundo y que no sea descabellado pensar en llevar a cabo técnicas de eliminación selectiva de armónicos en tiempo real. Uno de estos potentes sistemas de procesamiento digital es el sistema denominado CUDA, que presenta una capacidad de cálculo muy alta gracias al múltiple procesamiento paralelo de instrucciones. Un diagrama de la arquitectura se muestra en Fig. 2. Este tipo de sistema hardware se ha diseñado inicialmente para el procesamiento de gráficos y texturas, pero su diseño versátil ha permitido también modelos dedicados a las tareas específicas del ámbito científico.

El alcance de este trabajo se fijará en dos hitos relacionados con el problema de mitigación selectiva de armónicos:

1. El estudio de un método existente para la resolución del problema de SHM, conocido por *simmulated annealing*. Para ello se partirá de la existente implementación del



Odd harmonics				Even Harmonics	
Not triplen		Triplen			
Harmonic order (n)	Relative Voltage (Li)	Harmonic order (n)	Relative Voltage (Li)	Harmonic order (n)	Relative Voltage(Li)
5	6%	3	5%	2	2%
7	5%	9	1.5%	4	1%
11	3.5%	15	0,5%	6...10	0.5%
13	3%	21	0.5%	>10	0.2%
17	2%	>21	0.2%		
19	1.5%				
23	1.5%				
25	1.5%				
>25	0.2+32.5/n				

Figure 1 Implementación de un algoritmo de eliminación selectiva de armónicos de tres ángulos y necesidad de realizar una mitigación de armónicos para cumplir con la normativa.

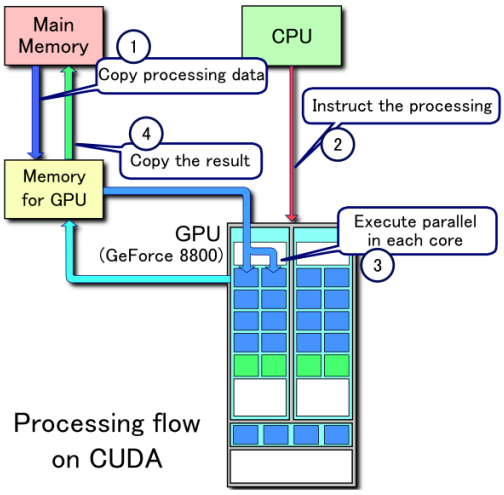


Figure 2 Procesamiento paralelo de datos en el sistema CUDA. Reproducido bajo licencia CC BY 3.0 por *Tosaka-commonswiki*.

algoritmo en CUDA, sobre la cual se estudiarán las posibilidades de optimización que ofrecen las nuevas herramientas de desarrollo disponibles.

2. La propuesta de un nuevo algoritmo, propuesto en 2014 por Naser Ghorbani y Ebrahim Babaeri, conocido por *exchange market algorithm* (*EMA*). Se realizará la implementación del problema del mismo en el entorno de MATLAB, comparándose los tiempos de ejecución de ambas resoluciones. Se realizará un montaje en Simulink/SimPowerSystems con motivo de probar las soluciones obtenidas. Por último, se implementará un interfaz sencillo, con la finalidad de facilitar al usuario la interacción con el generador de soluciones.

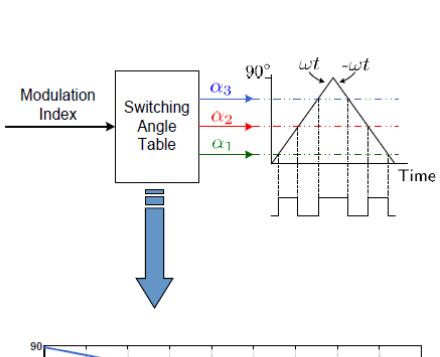
Realizada el estudio cuantitativo, se mostrarán algunas nociones sobre las posibilidad de implementación en tiempo real (*online*) del problema de mitigación selectiva de armónicos.

Abstract

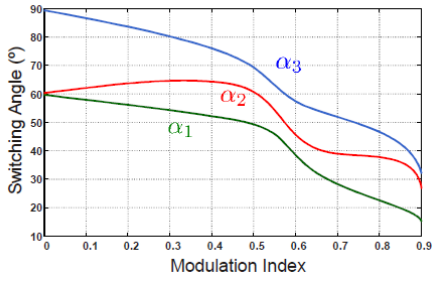
The actual increasing demand on energy has boosted the production of high-power production systems. However, these systems requires lower tolerances related their efficiency, robustness, fault tolerances, easiness of maintenance, etc.

One of the algorithms used for controlling these systems is selective harmonic elimination. This control method and modulation is based on the study of the theoretical output waveform and its harmonics content. This leads to easier output filter design and limiting switching loses, allowing the performance to reach compliance with grid normative.

Selective harmonic elimination requires enormous computing capabilities, due to the complexity of equation systems, which involves a high number of nonlinear operations. Differents methods have already been proposed (as shown in Fig. 3), although their complexity still requires those methods to be executed *offline*. Besides, real-time methods could improve converter performance and efficiency.



Odd harmonics				Even Harmonics	
Not triplen		Triplen			
Harmonic order (n)	Relative Voltage (Li)	Harmonic order (n)	Relative Voltage (Li)	Harmonic order (n)	Relative Voltage(Li)
5	6%	3	5%	2	2%
7	5%	9	1.5%	4	1%
11	3.5%	15	0,5%	6...10	0.5%
13	3%	21	0.5%	>10	0.2%
17	2%	>21	0.2%		
19	1.5%				
23	1.5%				
25	1.5%				
>25	0.2+32.5/n				



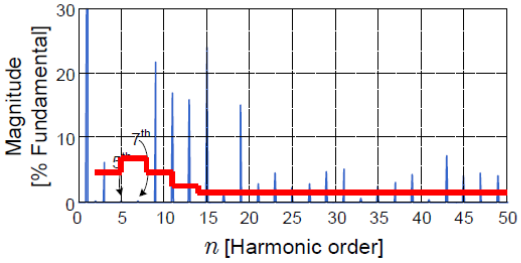


Figure 3 Implementation of an algorithm for selective harmonic elimination with three switching angles and the need for mitigation algorithm for grid normative compliance.

Moreover, recently-proposed technique selective harmonic mitigation aims to solve a inequalities systems, increasing the computational difficulty.

During the late years, the develope of new microprocessing systems have made possible performaces above millions of instructions per seconds, which rise a hope for real-time implementations of these algorithms. One of those powerful digital processing systems is CUDA, which exploits the capabilites of parallel execution, as shown in Fig. 4. Even though this hardware structure was originally design fo graphics and texture processing, its convenient arrangement has allowed devices for solely scientific purposes.

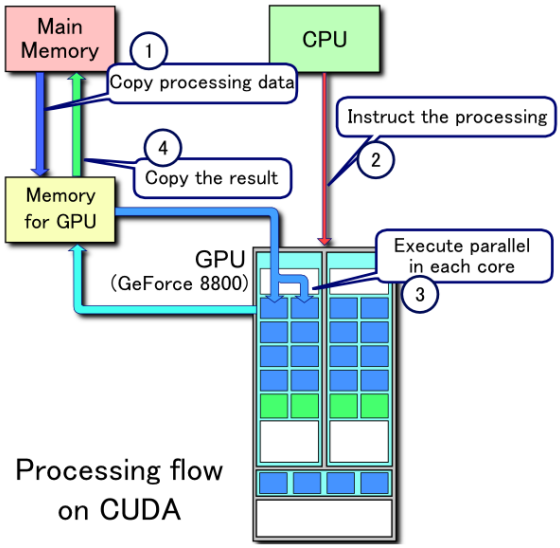


Figure 4 Parallel data processing on CUDA. Reproduced under CC BY 3.0 by *Tosaka-commonswiki*.

The scope of this work focus on two milestones:

1. The review of an existing method for solving the SHM problem, known as *simmulated annealing*. This research line aims on existing implementation on CUDA, in order to improve its execution time thanks to newer available developing tools.
2. The proposal of a newer algorithm, published for first time in 2014 by Naser Ghorbani and Ebrahim Babaeri, known as *exchange market algorithm* (EMA). An implementation of the algorithm in MATLAB, in order to compare execution time of both algorithm, is studied. In order to study the output solution, an experimental setup is going to be implemented in Simulink/SimPowerSystems. Lastly, a simple user interface is going to be developed in other to ease the generation of switching angles for the user.

After the quantitative study, a review of the possibility of a online implementation of SHM is presented.

Contents

Acronyms

ABC	Artificial Bee Colony Algorithm
AC	Alternating Current
ACO	Ant Colony Optimization
B2B	Back-To-Back
CHB	Cascade H-Bridge
DC	Direct Current
EMA	Exchange Market Algorithm
FACTS	Flexible AC Transmission Systems
FC	Flying Capacitor
GPU	Graphics Processing Unit
GSA	Gravitational Search Algorithm
GTO	Gate-Turnoff Thyristor
HVDC	High Voltage DC
IGBT	Insulated-Gate Bipolar Transistor
m_a	Modulation Index
MOSFET	Metal-oxide-semiconductor Field-Efect Transistor
OF	Objective function
PSO	Particle Swarm Optimization
PV	Photovoltaic
PWM	Pulse Width Modulation
SA	Simulated Annealing
SHE	Selective Harmonic Elimination
SHM	Selective Harmonic Mitigation
THD	Total Harmonic Distorsion

Acrónimos

ABC	Algoritmo de la colonia de abejas artificial
AC	Corriente alterna
ACO	Algoritmo de la colonia de hormigas
B2B	Back-To-Back
CHB	Cascade H-Bridge
DC	Corriente contínua
EMA	Exchange Market Algorithm
FACTS	Sistemas de transmisión flexible en AC
FC	Flying Capacitor
GPU	Unidad de procesamiento de gráficos
GSA	Algoritmo de búsqueda gravitacional
GTO	Tiristor de apagado en puerta
HVDC	DC de Alto Voltaje
IGBT	Transistor bipolar de puerta aislada
m_a	Índice de modulación
MOSFET	Transistor de efecto campo de metal-óxido-semiconductor
OF	Función objetivo
PSO	Optimización por enjambre de partículas
PV	Fotovoltáica
PWM	Modulación en ancho de pulsos
SA	Recocido simulado
SHE	Eliminación selectiva de armónicos
SHM	Mitigación selectiva de armónicos
THD	Distorsión armónica total

List of Figures

List of Tables

1 Introduction

The use nowadays of electronic power converters is key in our society, as very high voltage distribution methods and very high power equipment require a reliable, efficient and interactive supply of energy. As the use of traditional solutions run out of room for improvement, leading to an increase of cost, robustness and complexity to maintain required performance at such high power, these new devices opens a new way for reliable, efficient and adaptive devices.

On the present chapter, a review on power electronic converters will be made, specifically on the algorithms involved on this study and the techniques proposed for their mathematical computation.

Figure 1.1 ABB PVS800 power inverter, rated for 100-1000 kW, courtesy ABB.

1.1 Review of different power converters

Power converters can be classified in four groups, regarding its input and output [2].

- **DC/DC converters**, which supplies a DC output from a DC input. Non-isolated topologies, as for example *boost*, *buck*, *buck-boost* or *Cuk* topologies, as well as galvanic-isolated converter as *flyback* or *forward*, can be fitted in this category.

- **DC/AC converters**, which supplies an AC output from a DC input. These are commonly called as *inverters*. A further review on this will be done in the next section.
- **AC/DC converters**, which supplies a DC output from an AC input. These are commonly called as *rectifiers*, which can be implemented as uncontrolled -half and full wave diode rectifier for single or three phases applications- and controlled -thyristor based controlled for half or full wave.
- **AC/AC converters**, which supplies an AC output from an AC input. One possible denomination for this group is *matrix converters*, which are normally designed as the combination of a AC/DC converter and DC/AC inverter, connected through a DC-link stage. Topologies as *back-to-back* (B2B) are proposed in other to ease the control actions and provide capabilities for bidirectional power flow, which can be useful for motor drive application on which regenerative braking is present.

Power converter can be implemented in a wide scope of application, ranging from simple domestic implementation - line adapters, battery charging [3], domestic photovoltaic (PV) systems- to higher load application -motor drive for industry and transportation, industrial process control [4] [5]- and electricity distribution infrastructure -grid adaptation for different energy sources, as solar or wind [6], grid interconnection, as established between Spain and France or Peninsular Spain and Balearic Islands [7], power flow and reactive power compensation as implemented in FACTS systems [8]. Not only devices have to be specifically adapted for withstanding high power level, but topologies and control methods have to be chosen so those losses are minimized while ensuring safety and performance specifications.

Transmission technologies present also a variety of presentation of electric power: from AC transport and distribution, both in single and three phases, to newly proposed solutions relying on DC connection on high voltage levels (HVDC). While three-phases AC is the most worldwide-spread method for wide area transport, due to its best performance on terms of line looses, sizing of manoeuvring devices and lines, and industrial applications, single-phase AC is also relevant due to domestic use. HVDC is nowadays on increase in order to implement connection between grids where capacitive and inductive effects redeem AC distribution unsuitable, as for example on submarine lines [9].

The scope of this study will be three-phase power inverters. The use of these systems is determinant in numerous environments. For example solar arrays, which output a DC current dependant of light irradiance, require an adaptive power inverter for supplying residential users at a stable level in AC or for interconnecting it to mains distribution grids.

The increase of power sources that can be exploited nowadays widens the way that electricity is presented. While increasing complexity and control variables, the expected outcome in terms on efficiency and sustainability of a interconnected matrix of generation points and consumers, give birth to the concept of smart grid, which is one of the most populated research nests nowadays

1.1.1 Concept of *smart grid*

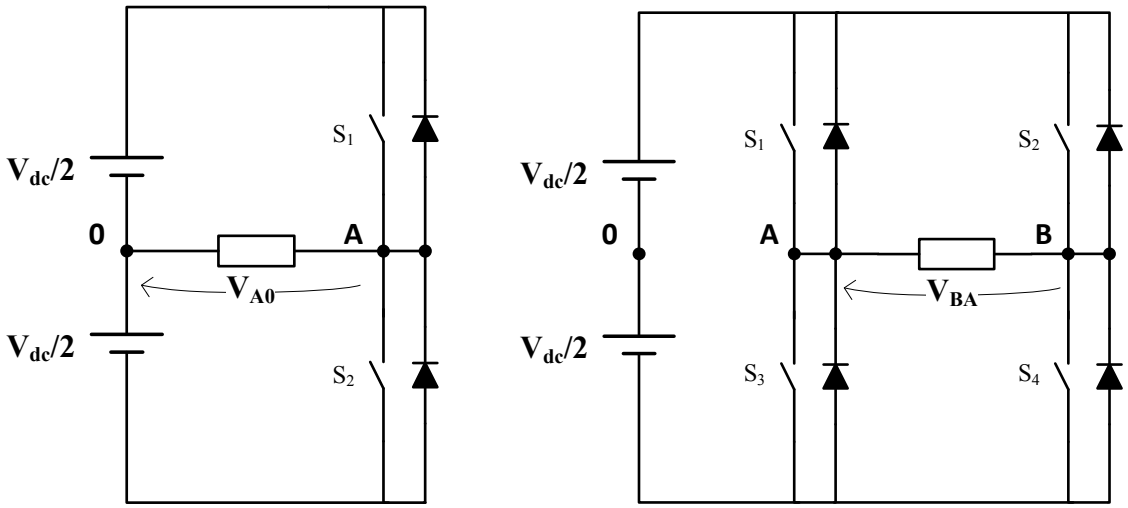
Smart grid systems aim to an interconnected power distribution system, where power flows can be set arbitrary according to users requirements and source generation [10]. On a conventional distribution system, energy flows occur straight from generation sources

to consumers, with generation controlled by current demand. This control scheme is incompatible with some alternative power sources, for example solar and wind energy, with a production dependant on time of day, leading to a overproduction during periods that cannot be fully exploited. Smart grids aims to reach an interconnected net of consumers and generation points, where the flows of energy may adjust to increase the performance of the whole system, increasing the relevance of alternative sources and using all windows of opportunity, including the storage and supply of that overproduction [11].

In other to reach this concept, power electronics is due to achieve new milestones in electronics, control theory, communications, etc [12].

1.2 Topology and modulation of power inverters

The easiest design for an inverter is the bipolar inverter, so that output voltage can vary only between a positive level and its opposite, as shown in Fig. 1.2 . This can be implemented by use of half-bridge or full-bridge topology, as seen in Fig. 1.3. Solid state devices provide varies between ON and OFF states, therefore behaving as switches.



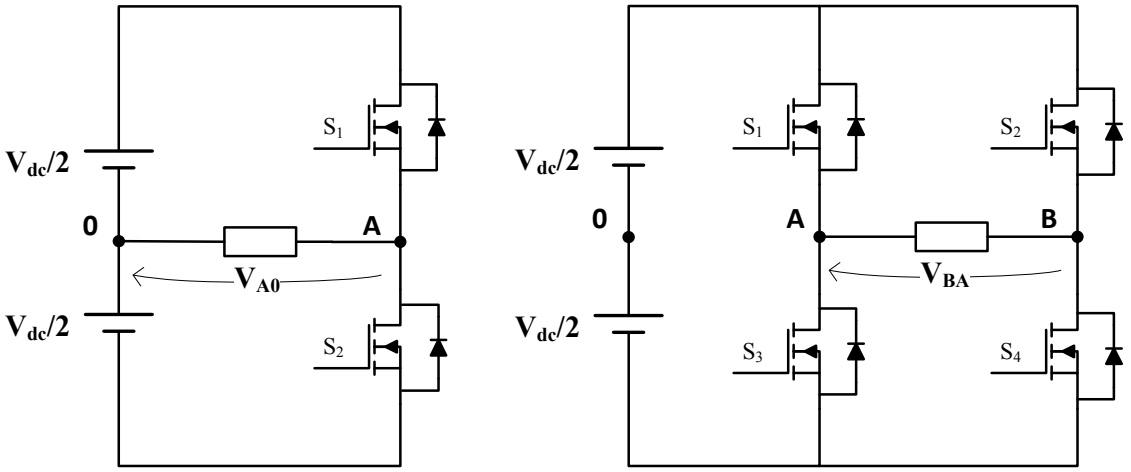
(a) Two level. **(b)** Three level.

Figure 1.2 Principles of bipolar power inverters.

However, full-bridge topology offers the possibility of accessing for a third level, a zero voltage level. Better performance, in terms of harmonics content and distortion, can be therefore achieved.

These two topologies forms the basic of bipolar inverters. Several control methods have been proposed for these topologies: for example, bipolar PWM for half and full bridge topologies, unipolar PWM for full-bridge topologies, selective harmonic elimination/mitigation, space-vector modulation for three-phases implementations, etc.

The typical constitution these power inverters relays on solid state switching devices, as MOSFET for low power, IGBT for medium power and GTO for higher current. The output waveform approximates the sinusoidal voltage by transitions between discrete voltage level, resulting in an important amount of harmonic distortion [13].



(a) Two level. **(b)** Three level.

Figure 1.3 Implementations of bipolar power inverters.

As a consequence, filtering is required in subsequently stages, which increases the losses of the converter, the size -as coils or capacitors become bulkier as voltages and currents increase- and price -for example, coils for handling high current and high voltages requires larger surfaces for current flow and more resistant insulation.

Two paths have been followed in order to maintain the sustainability of power inverters facing the increase on power delivery nowadays:

- Modulations becomes more complex, normally by means of precomputation of output waveform harmonic content and optimization applied on the commutations. This results in the increase of the complexity of computation, thus resulting in less flexibility in real time.
- Increase of the number of commutations, i.e. increasing the switching frequency of the power converter. By this, harmonic content is shifted to higher frequencies, which results in lower values of capacitance and inductance required in filters and smaller and cheaper solutions. However, this is limited by restriction presents in response time of semiconductors, control logic and dead times imposed by the topology.

In order to overcome the stiffness of bipolar inverter, by combination of the two and three level topologies, new arrangement have been proposed to overcome the limitation for wider ranges of voltages: *flying capacitor* (FC), *neutral point clapped* and *cascade H-bridge* (CHB) topologies are the most relevant for multilevel inverters [5].

Control techniques for multilevel inverter evolve for those designed for bipolar case: phase-shifted PWM and level-shifted PWM extend the bipolar and unipolar PWM by addition of a higher number of carrier waves; Multilevel Space Vector modulation and nearest-vector expand the $\alpha\beta$ representation of levels, and staircase modulation applies the principles of SHE [14].

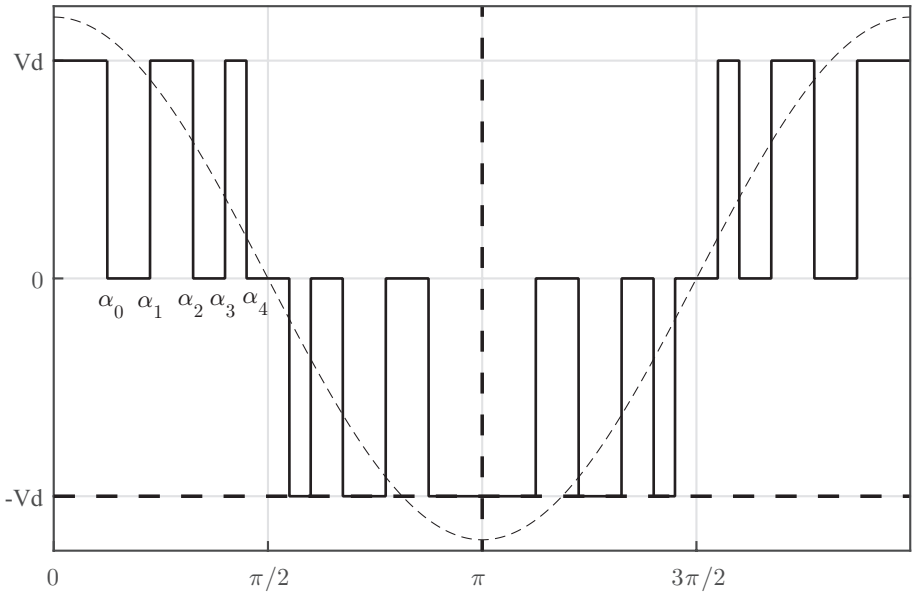


Figure 1.4 Waveform pattern for five switching angles.

1.3 SHE and SHM algorithms

Selective harmonic mitigation expands the present idea of selective harmonic elimination method, adding a higher degree of specify output harmonic content by means of increasing the computational complexity of the problem. Both methods focus on the theoretical shape of output waveform, as a PWM output wave consisting on k switching angles for the first quarter of cycle. In order to maintain the output waveform symmetric, a cosine symmetry condition can be specified, and thus application time of the switching angles is defined for the full period. The resultant waveform is represented in Fig. 1.4

Using the Fourier transform of that wave, the amplitude for the j^{th} order harmonic is estimated as:

$$H_j = \frac{4}{j\pi} \sum_{i=0}^{k-1} [(-1)^i sin(j\,\alpha_i)] \tag{1.1}$$

Where $j = 1, 2, ...n$ and $[\alpha_0, \alpha_1, ..., \alpha_{k-1}]$ be the switching angles.

However, applying several design consideration can reduce the numbers of harmonics into consideration, for example:

- **Using a symmetrical waveform**, which zeroes even harmonics.
- **Considering a three-phase without neutral topology**, which cancels triplen harmonics in line-to-line voltages.

For the following formulation, it will be considered a three-level inverter for a basic three-phase application, due to the benefits in term of harmonic content under balanced loads.

On selective harmonic elimination, by application of N switching angles, it is allowed to choose arbitrary the value of N harmonics, which is often formulated for driving N-1 first odd non-triplen harmonics to zero and fixing the value of the main harmonic to a desired

value [15] [16]. Therefore, the equations in 1.1 can be reformulated as the target system of equations:

$$\begin{aligned} m_a &= \frac{4}{\pi}\sum_{i=0}^{N-1}[(-1)^i sin(\alpha_i)] \\ 0 &= \frac{4}{j\pi}\sum_{i=0}^{N-1}[(-1)^i sin(j\,\alpha_i)] \\ &\text{where } j = 5,7,11,...,L \end{aligned} \tag{1.2}$$

The computational difficulty of 1.2 is related to the nonlinearity of the equations forming the system due to the presence of *sin* function, which appears N^2 times for SHE case. Several methods for solving the systems are documented, being Newton-Raphson usually proposed. Newer methods, for example relying in genetic algorithms, has overcome many of the drawback for fast computation [17]

A pseudocode for this method is proposed below:

1. Generate an initial iterand, within range $0 \leq \alpha_0 \leq ... \leq \alpha_{N-1} \leq \frac{\pi}{2}$

2. Being $x = [\alpha_0, ...\alpha_{N-1}]$, the function $\mathbf{f}(x)$ is created by evaluating the terms in 1.2 as:

$$\mathbf{f}(x) = \begin{bmatrix} H_1 - \frac{4}{\pi}\sum_{i=0}^{N-1}[(-1)^i sin(\alpha_i)] \\ \vdots \\ \frac{4}{j\pi}\sum_{i=0}^{N-1}[(-1)^i sin(j\alpha_i)] \\ \vdots \end{bmatrix}$$

and its jacobian $J(x)$,

$$J(x) = \left[\frac{\partial \mathbf{f}}{\partial \alpha_0}, \ldots, \frac{\partial \mathbf{f}}{\partial \alpha_i}, \ldots\right]$$

3. The error $\mathbf{e}$ is evaluated as the solution of $J(x)\mathbf{e} = \mathbf{x}$

4. Until a required tolerance is not reach, $|e| = TOL_r|x| + TOL_a$:

a) The proposed solution $x^* = x - e$ is taken.

b) Error is evaluated as in **3.** and conditional expression in **4.** decides the program flow.

The example represented in Fig. 1.5 show the computed switching angles values by application Newton-Raphson method for a system with 5 switching angles. Initial iterands were choosen by uniformly distributing five angles in range $[0°, 45°]$

However, the instability of this method related to its initial iterand [18], which may lead the system to out-of-range solutions, and the computational difficulty of evaluating the jacobian matrix of the system in 1.2, makes it a poor solution for time-concerned applications.

SHM was proposed in 2007 in [19], in order to expands the capabilities of SHE to a higher level due to the increasing power of calculation and optimization algorithm. In contrast to SHE, in SHM the condition of setting the N^{th} first harmonics to zero is relaxed, letting them be equal or smaller than a certain value, which must be set according to the

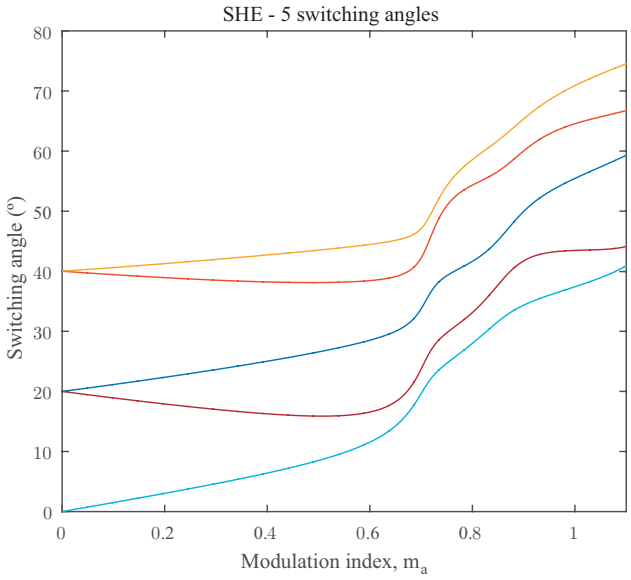


Figure 1.5 Solution for m_a using Newton-Ralpson for 5 switching angles.

suitable grid code. Therefore, the system of equations defined in 1.2 turns into a system of inequalities, so that the number of harmonics to be considered can be greater than N. That allows the designer to archive different targets, for example, considering the total harmonic contents of the output waveform, so called THD, for a greater number of harmonics than already being mitigated [20] [21]. Other possible consideration is cost of output filter, which may suppose an important cut on filtering costs in very high power applications.

For the following, N switching angles and M harmonics will be considered, including main harmonic, which will be set, within a tolerance on the parameter m_a. The limit for the harmonic can be define as a vector $\mathbf{L} = [L_1, ...L_M]$, so that:

$$\begin{aligned} |m_a - H_1| = |m_a - \frac{4}{\pi}\sum_{i=0}^{N-1}[(-1)^i sin(\alpha_i)]| \leq L_1 \\ \frac{4}{j\pi}\sum_{i=0}^{N-1}[(-1)^i sin(j\alpha_i)] \leq L_j \end{aligned} \tag{1.3}$$

where $j = 5,7,11,...M^{th}$ non-triplen odd harmonic.

The system of equation can be expressed, however, as a unidimensional function, so called **objective function** of the problem. For this function, it is necessary to define the error variables, which measure how far the j^{th} harmonic is for the limit:

$$\begin{aligned} E_1 = |m_a - H_1| - L_1 \\ E_j = H_j - L_j \end{aligned} \tag{1.4}$$

And objective function can be define as:

$$f(\alpha_0, ..., \alpha_{N-1}) = \sum_{j=1}^{M} c_j E_j \tag{1.5}$$

Where c_j are set according a strategy considering if the harmonic content is below the limit: if $|H_j| > L_j$, then $c_j = 1000$, otherwise $c_j = 1$.

So far, using SHM, the following goals can be archived simultaneously:

- $M \geq N$ harmonics can be set below the grid limits
- More general parameters can be considered into calculation, for example THD for a larger number of harmonics
- Parameters not directly related to harmonics, as for example output filter cost, can be considered
- As the strong restriction imposed by SHE are relaxed in SHM, solution to some problems is now allowed.

As an example of grid code, international standard EN 50160 [22] and the quality reference established in QUALITY CIGRE WG 36-05 is represented in table 1.1

Table 1.1 GRID CODE EN 50160 + QUALITY CIGRE WG 36-05 restrictions.

Odd harmonics				Even harmonics	
Not triplen		Triplen			
Harmonic order	Relative voltage L_j	Harmonic order	Relative voltage L_j	Harmonic order	Relative voltage L_j
5	6%	3	5%	2	2%
7	5%	9	1.5%	4	1%
11	3.5%	15	0.5%	6...10	0.5%
13	3%	21	0.5%	>10	0.2%
17	2%	>21	0.2%		
19	1.5%				
23	1.5%				
25	1.5%				
>25	0.2+32.5/n				

However, the main downside of the method is its complexity, which only allows heuristic methods for finding a solution. These methods are mainly iterative, tending to be high time-consuming because of large number of evaluation of the objective function. Adding the large number of computation of trigonometric functions, programming paradigm and optimization techniques have to be choose thoroughly to ensure code performance.

1.4 Methods of solving

For the solution of nonlinear system of equation, several metaheuristic have been documented. A metaheuristic problem provides a solution to optimization by partially sampling a domain which can be considered impossible to completely analysis within a time frame [23]. In a evolutionary algorithm, the sampling is inspired in the procedures of recombination, mutation, reproduction, etcetera, of a set of solutions, normally called a population. These algorithms' behavior is normally inspired in physics or natural phenomena, which acts as a

concept proof of its performance and convergence, for example, ants pheromone-driven search for food, solidification of a metal, etc.

Metaheuristic perform the search of a local or global minima from a wide set of solutions, basing its convergence in probabilistic terms, as a large number of possible solutions are tested. This can be compared to *gradient based* methods which relies on the gradient of the objective function. Other possible classification relies on how the candidate points for solution are arranged: on **single-solution methods**, every iteration one point is modified so that it continues approaching to the local or global minima. On **population-based methods**, however, every iteration works over a set of points, which are combined to each other in order to increase the fitness of these points to the objective function. Fitness of the points is evaluated in parallelism with a natural of physical magnitude which behaviour is being mocked up, for example the energy or velocity of particles, etc.

Some of these algorithms are shown below:

- **Particle swarm optimization (PSO)**: Correlates the solution points with a swarm of birds or insects, finding the next set of points applying equations that mock the velocity and position of elements in the swarm, and converging into the global minima [24]
- **Artificial bee colony algorithm (ABC)**: Provides an global minima of the objective function mimicking the hierarchy relations on a swarm of bees: *employed bees* marks the position of *food*, *scout bees* looks around for new sources of food and *onlookers bees* decide which source to keep active or to start new search [25]
- **Ant colony optimization (ACO)**: A first population of ants walks randomly over the solution space upon finding *food* leaving a *pheromone trail*. Pheromone trail is followed by new generations, with the possibility of finding shorter paths as the trails fade out, which avoid getting stuck on local minima [26]
- **Tabu search**: The global minima is found by moving iteratively from *low score* local minima, which are listed -*tabu list* and avoided in forward iteration of the algorithm, which increase of calculation [27].
- **Genetic algorithm**: An initial population of solution is governed by rules: 1st. selection, which eliminates worst solutions, 2nd. crossover, which combines characteristic of two of these solutions and, 3rd. mutation, which adds new characteristic to some of final solutions. Letting the population to evolve for a sufficient number of iterations, it will approach to the local minima of the objetive function. [17]

Two of these algorithms, *Exchange Market Algorithm* and *Simulated Annealing*, will be described specifically on chapters 4 and 5 respectively, due to their importance on this study.

2 Introducción

El uso actual de la electrónica de potencia ha crecido gracias a las capacidades de ésta de proporcionar métodos de distribución más fiables, eficientes y con mayores posibilidades de actuación. Esto supone un fuerte contraste con soluciones anteriores, en las que el uso de tecnologías tradicionales llevaba a un incremento en coste, tamaño y dificultad para matener los niveles de alta potencia deseados.

El uso de los convertidores de potencia -en sus formas de DC/DC, DC/AC, AC/DC y AC/AC- cubre un gran número de necesidades actuales, desde el ámbito de aplicaciones domésticas -adaptadores, cargadores de baterías [3], sistemas fotovoltáicos - hasta aplicaciones de mayores potencias -control de motores para aplicaciones industriales y de transporte, equipos industriales de alta potencia [4] [5] - así como para labores de transporte y distribución - adaptación de fuentes de energía a la red, como la eólica y la solar [6], interconexión de redes, como la establecidad entre España y Francia o entre España Peninsular y las Islas Baleares [7]. o equipos de control de flujo de potencia y de potencia reactiva, como los implementados bajo la denominación FACTS [8].

En el ámbito de transmisión, la variedad existente en métodos lleva a distintos objetivos: distribución y transporte en alterna trifásica, debido a las bajas pérdidas en líneas largas aéreas, distribución a nivel de usuario mediante instalaciones monofásicas en alterna, debido a la estandarización de niveles de tensión, así como nuevas propuestas de transmisión en DC. Estas nuevas técnicas se enmarcan dentro de las tecnologías HVDC, las cuales se implementan para el transporte cuando no es viable la trasmisión estándar, por ejemplo, por diferencia de niveles o por uso de líneas subterráneas o acuáticas, donde los efectos capacitivos hacen la línea inviable [9].

El objetivo de este estudio serán los inversores trifásicos. El uso de inversores es, a día de hoy, esencial en un gran número de aplicaciones. Por ejemplo, para la conexión de paneles solares, cuya salida es una corriente contínua dependiente de la irradiancia solar, es necesario implementar etapas de inversores, así como convertidores DC/DC, para proporcionar estabilidad y niveles de tensión adecuados para la interconexión con la red [11].

2.1 Topologías y modulaciones

La implementación más simple de un inversor es la de dos y tres niveles [2]. En esta topología, varios dispositivos semiconductores permiten disponer de una tensión de salida

que puede ser la tensión máxima y su opuesta, en el caso de dos niveles, y entre éstas y cero, en el caso de tres niveles. Estos convertidores pueden montarse mediante topologías de medio puente (*half bridge*), en el caso de dos niveles, así como de puente completo, en el caso de dos y tres nivele. Una representación del principio de funcionamiento de estos convertidores, así como el montaje con ambas topologías, puede verse en Fig. 1.2 y Fig 1.3, respectivamente.

Para su control, se han desarrollado un número considerable de aplicaciones, ya sea para su implementación en monofásica como en aplicaciones trifásicas. Podemos destacar: PWM bipolar, para uso en dos niveles, PWM unipolar, para uso en tres niveles, *space vector modulation* (SVM), para uso en inversores trifásicos, y la eliminación y mitigación selectiva de armónicos (SHE y SHM).

Para mejorar el contenido espectral de la tensión de salida, y con vistas a disminuir las etapas de filtrado necesarias -formadas por bobinas y condensadores de tamaños, peso y coste elevado- se han implementado topologías que permiten disponer de un número mayor de tensiones, de forma que se reduzcan las necesidades de filtrado de armónicos de mayor frecuencia fácilmente eliminable con filtros más pequeños. De esta forma, se han propuesto tres topologías principales de inversores multinivel [5]: *flying capacitor*, *neutral point clapped* y *cascade H bridge*. Los métodos propuestos para su control son adaptaciones a los realizados para aplicaciones unipolares y bipolares, por ejemplo: level-shifted PWM y phase-shifted PWM, basado en PWM unipolar y bipolar, multilevel SVM y nearest vector modulation, ampliando la distribución de niveles en el plano $\alpha\beta$, y modulación en escalera (staircase modulation), como implementación de los algoritmos SHE/SHM [14].

2.2 Problemas SHE y SHM

Centrándonos en el problema de eliminación de armónicos, el algoritmo parte del cálculo del contenido armónico presente en un cuarto de ciclo de la tensión de salida, en función de un número k de ángulos de corte. Con motivo de mantener la señal simétrica, la reconstrucción de la misma para el periodo completo puede obtenerse mediante la aplicación de simetrías de tipo coseno, quedando completamente definida los tiempos de aplicación de los ángulos de corte. Esta distribución puede verse en 1.4.

El contenido del armónico $j-simo$ puede calcularse empleando el desarrollo en series de Fourier, de la siguiente forma:

$$H_j = \frac{4}{j\pi} \sum_{i=0}^{k-1} [(-1)^i sin(j\alpha_i)] \tag{2.1}$$

Donde $j = 1,2,...n$ y $[\alpha_0, \alpha_1, ..., \alpha_{k-1}]$ se definen como los ángulos de corte.

Al haberse elegido k ángulos de corte, se verifica que puede fijarse arbitrariamente el valor de k armónicos. Suponiendo que se va a emplear en una aplicación trífasica, se sabe que, en magnitudes de línea los armónicos triples serán nulos lo cual, unido a usar una señales simétricas, permite reducir los ármonicos a considerar a los armónicos impares no múltiplos de 3 [21].

Generalmente, el problema SHM se plantea para anular los $k-1$ primeros armónicos, fijando el valor del armónico fundamental a un valor m_a conocido, definido como índice

de modulación, de la siguiente forma:

$$\begin{aligned} m_a &= \frac{4}{\pi}\sum_{i=0}^{N-1}[(-1)^i sin(\alpha_i)] \\ 0 &= \frac{4}{j\pi}\sum_{i=0}^{N-1}[(-1)^i sin(j\alpha_i)] \\ &\text{donde } j = 5,7,11,...L \end{aligned} \tag{2.2}$$

Este problema que es resoluble mediante métodos de resoluciones de sistemas no lineales, siendo el más conocido de ellos Newtown-Ralpson. Debido a la fuerte dependencia de la solución con el iterando inicial, así como la imposibilidad de establecer condiciones sobre armónicos de orden superior -los cuales en general alcanzan valores no aceptables- obligan a su uso exclusivo en aplicaciones de potencias bajas.

Para evitar este problema, se propuso en 2007 el método de mitigación selectiva de armónicos (SHM). Este problema formula las restricciones establecidas en 2.2 en forma de inecuación, es decir, establece un límite superior al valor de cada armónico. De esta forma, no sólo es posible considerar más armónicos en el problema que ángulos de corte, sino que, al relajar la condiciones individuales sobre cada armónico, el contenido esperado en armónicos de orden superior se ve reducido al obtenido en el caso de SHE [20] [15] [16].

Aplicando estos conceptos a lo establecido en 2.2, se llega a las siguientes expresiones:

$$\begin{aligned} |m_a - H_1| = |m_a - \frac{4}{\pi}\sum_{i=0}^{N-1}[(-1)^i sin(\alpha_i)]| \leq L_1 \\ \frac{4}{j\pi}\sum_{i=0}^{N-1}[(-1)^i sin(j\alpha_i)] \leq L_j \end{aligned} \tag{2.3}$$

donde $j = 5,7,11,...$ M-ésimo armónico impar no múltiplo de 3.

Lo cual puede reescribirse en forma de errores como:

$$\begin{aligned} E_1 = |m_a - H_1| - L_1 \\ E_j = H_j - L_j \end{aligned} \tag{2.4}$$

E incorporarse estos términos a una función de costes de la forma:

$$f(\alpha_0,...,\alpha_{N-1}) = \sum_{j=1}^{M} c_j E_j \tag{2.5}$$

Donde c_j se decide en función de la consecución del objetivo del armónico $j-simo$, como 1 si se ha conseguido el objetivo y un valor grande (por ejemplo, 1000) en caso contrario. Los límites de aplicación se elegirán de acuerdo a un *grid code*, por ejemplo bajo la normativa EN 50160 [22] y los estándares de calidad QUALITY CIGRE WG 36-05 representados en Tabla 2.1

Debido a la dificultad de las ecuaciones presentes, estos problemas se encuadran dentro de métodos metaheurísticos y álgoritmos evolutivos. Por ejemplo, algunos de estos métodos serían los algoritmos genéticos [17], *particle swarm* [24], *simmulated annealing* [28] o *exchange market algorithm* [29].

Table 2.1 GRID CODE EN 50160 + QUALITY CIGRE WG 36-05 restricciones.

Armónico impar				Armónico par	
No triple		Triple			
Orden del armónico	Tension relativa L_j	Orden del armónico	Tension relativa L_j	Orden del armónico	Tension relativa L_j
5	6%	3	5%	2	2%
7	5%	9	1.5%	4	1%
11	3.5%	15	0.5%	6…10	0.5%
13	3%	21	0.5%	>10	0.2%
17	2%	>21	0.2%		
19	1.5%				
23	1.5%				
25	1.5%				
>25	0.2+32.5/n				

3 Description of the tools used for code developing

The reason for this chapter is to provide to the reader an overview on the computation technologies which have been used to obtain the results along the document. Two environments have been especially relevant for this research. Firstly, results presented for selective annealing have been obtained under the CUDA platform. CUDA provides the user a fast developing method for parallel code execution, by exploiting the capabilities of graphic processing. Graphics processing has proven over the years its capabilities for handling elevated number of calculation on large blocks of data [30].

On the other side, for the development of Exchange Market Algorithm, MATLAB software has been used. MATLAB provides a programming language easy to learn and capable for implementing complex problems. In addition, a vast collection of tools, from a wide number of knowledge fields, is provided. Thanks to these factors, MATLAB has become a referent on Academia, with over five thousand Universities having use of it [31].

3.1 CUDA architecture and processing flow

The architecture of CUDA relies on vast hardware management while keeping user programming abstracted thanks to the API in programming language. In order to reach that paradigm, the solution was to establish two separate hierarchy: a *CUDA hierarchy* and a *GPU hierarchy* [32]. For programming, developers must work on CUDA field, while GPU mapping is done by compiler. So that, resource mapping on the GPU can be dismissed on a first approach to programming, while some understanding is requiered to ensure best optimization.

From CUDA point of view, the smaller unit of execution found is the **thread**. Threads constitute an instance of the kernel -the GPU instruction program- which can run as a single processor, with its own program counter, registers and memory. Several threads can be grouped on a higher structure called a **thread block** or, simpler, a **block**. While being a pack of threads, blocks offer a distinctive feature, *shared memory*. Shared memory is available for read/write operations for all threads in the same block. This feature is key for allowing cooperative processing and effective speedup [1].

The upper structure in this abstraction is the *grid*: a grid is defined as a group of blocks which executes on the same graphic board. One or more grid can be launched on the same

board. Grids have also a globally available memory space for storing program related variables and constant. A representation of these terms can be found at Fig. 3.1.

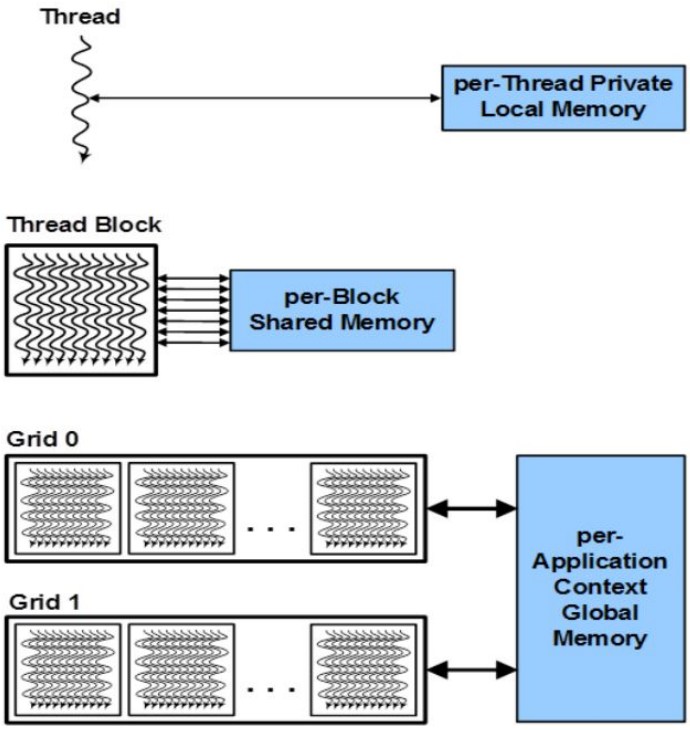


Figure 3.1 Thread, block and grid structures, as described from [1].

Thank to this memory architecture, variables can be associated with different operations: **global variables**, used to store program variables, output results and execution inputs and constants; **shared variables**, used for operations over matrix of data on a block, as for example sorting or linear algebra, as a means of reducing the amount of transfer operation between host and device, and **local variables**, with represent values unique for the current execution thread.

The CUDA abstraction is mapped over the hardware structure on the board. The top level element on this abstraction presented on a CUDA board is one (or more) *streaming multiprocessors (SM)*, simply called as multiprocessors. Each multiprocessor features a number of **streaming processors (SP)**, which can be refereed as a CUDA core. A CUDA core is composed as a complete arithmetic-logic unit, implementing vector oriented operations to extend code performance [1].

The execution of threads is organized in packs of 32 threads, which composes a *wrap* [33]. A wrap schedules the execution of every thread within its scope, by storing the relevant (status, instruction) registers. The program flow is managed by the *wrap dispatcher* and the *instruction dispatch unit*, which are implemented on the core logic and execute the corresponding instruction.

To sum up, the mapping between both abstraction is: one or more grids are executed by a same graphic card; one or more blocks are executed by the same stream multiprocessor, and one or more threads sum up to a wrap and is executed on the core. As wraps size is

fixed to 32 threads, number of threads per blocks are recommended to be kept as close as 32 or multiples of it, in order to reduce the unused resources per wrap [32].

3.1.1 Programming and interfacing

The main resource for programming in CUDA is the **CUDA Toolkit**, an integrated installer with provides user automated installation and setup of a large diversity of tools for programming. As for today, NVIDIA provides, on its 8.0 version, support over x64 architecture for both Windows, Linux and Mac.

The basic tool for CUDA programming is located at the **NVIDA CUDA Compiler Driver** or **NVCC**. The compiler is required for perform the conversion from plain text source code to binary data, which can be execute by the device. As stated before, CUDA programming relies on C++ as substrate from language, however this is extensively complemented by device-specific macros and directives. Even more, standard distributions of CUDA API includes bundle-in libraries, which provides seamlessly compatibility for scientific and mathematics functions -FFT, matrix algebra, etc.

The most important library so far is the *CUDA Runtime API*, which provides the required functions for communication between host and device: device initialization, error handling, thread synchronization, execution control, data transfer between device and host memories, and application benchmarking. Another important library is *Thrust*, which provides a collection of highly optimized functions for vector based operations, as for example sorting and maximum and minimum element. Using Thrust over user-defined functions ensures that code would run on high utilization of the hardware resources without special configuration by the user.

After compilation, a **linking** process is required for joining the different objects created in the previous step. In NVCC, no external linker is required, as its configuration is provided by adding options to NVCC call. While code can be provided in a single *.cu* file, including both machines and device code, NVCC needs a separate compiler for dealing with non-CUDA code. The solution chosen is to compile host code in a separate step, generally done by calling standard tools, as for example GCC and G++ in Linux architectures and CL.exe in Windows.

Linked objects are finally generated onto an executable or library binary file. An executable file is a compiled code format, which can be executed directly in the machine. In Windows, they can be found as *.exe* files, while on Linux are shown as *.out* files. Libraries provides, in contrast, compiled code that must run from another application. Two different types can be distinguished: 'static library', if the library is compiled within the application and bundle with it, or 'dynamic library', in which library file is located at a particular directory and acceded by applications when needed. Typically, static libraries are distributed as *.lib*, while dynamic libraries are found as *.dll* in Windows environments and *.so* on Linux.

CUDA compiler comes alongside with graphical user interface, which provides full support to ease the developing process. Some capabilities provides by the interface can be: file hierarchy and explorer, quick access to functions for compiling, running, etc., code autocompletion, syntax highlighting, code indenting and refactoring, shorcuts to variables/function declarations, ...

Up to date, two different IDEs have been implemented for CUDA programming: for programming on Windows platforms, Microsoft commercial software Visual Studio was

adapted, with the install of a plug-in (Fig. 3.2). For Linux platform, a modified version of open-source software Eclipse has been developed, called *NVIDIA Nsight* (Fig. 3.3).

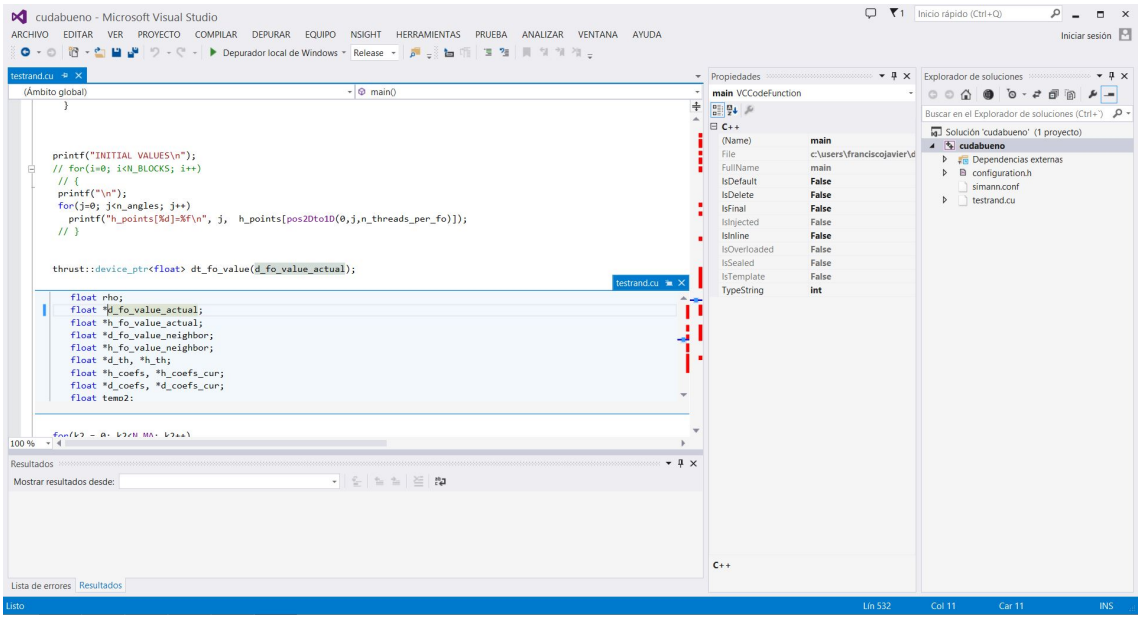


Figure 3.2 Example of Visual Studio IDE for CUDA programming, showing one of its features, declaration highlight.

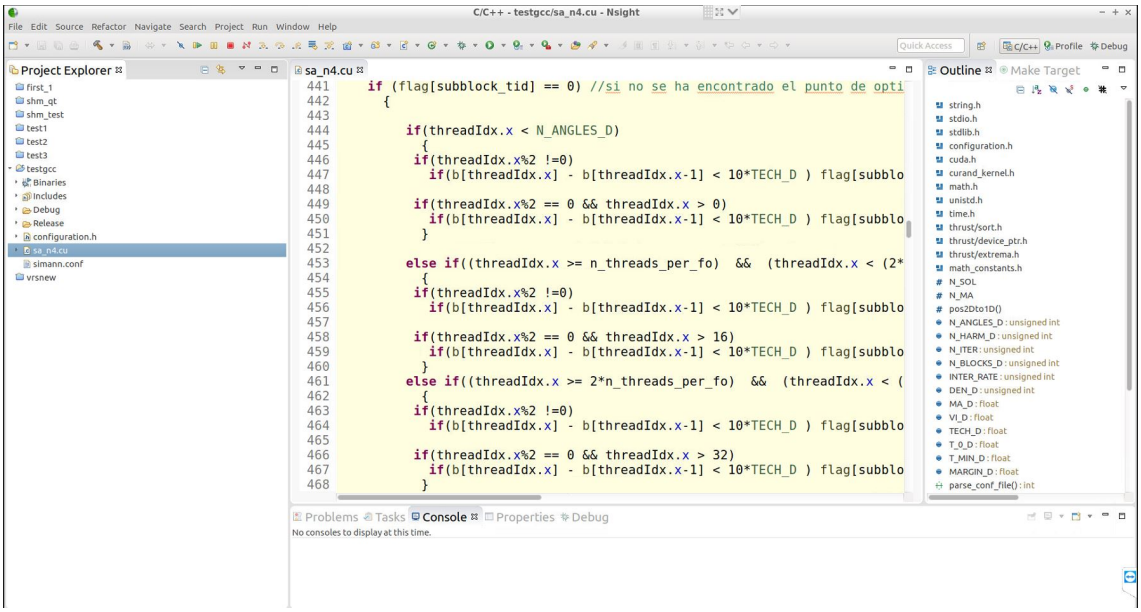


Figure 3.3 Example of NVIDA Nsight IDE.

Other important feature implemented on CUDA is debugging, which is provided by CUDA-GDB. This implementation provides basic features found on common debuggers, as for example step by step execution, disassembly, local variable scoping, and breakpointing, while other advanced tasks, as device variable access and block occupancy and status report.

Performance of processing is a critical task on CUDA architecture, as wrong synchronization can lead to an actual slow down in execution time. For this particular task, special tools have been implemented, being the most relevant the *NVIDIA Visual Profiler*. This tool provides, from a single window, several parameters related to optimization:

- Chronogram of execution of functions along the code.
- Optimization suggestions, as for example possible improvements in memory transfer operations.
- Optimization ranking, evaluating the feasibility of improvement on the function.
- Resource utilization, from the maximum available resource.
- Ideal parameter selection, in terms of blocks and threads.

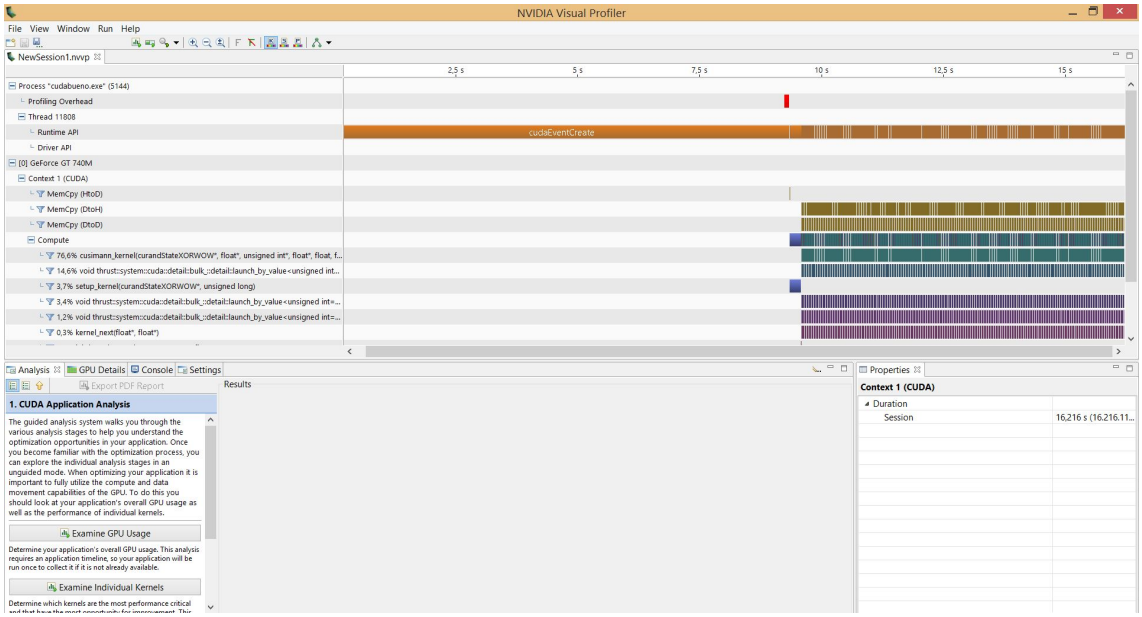


Figure 3.4 Example of NVIDA Visual Profiler.

Documentation for CUDA programming is also vastly available within the CUDA Toolkit. One of the main sources is the CUDA Toolkit Documentation, part of NVIDIA Developer Zone This resource provides a large database of function declarations for native and library functions, parallel processing workflow, recommended practises and installation and use guides. This guide is also available online, at [34]. In addition, a large collection of sample projects are provided at *CUDA Samples*. Several examples, covering from basic utilities as graphic card identification and bandwidth test to advanced fluids simulations are available. One example of them is represented in Fig. 3.5.

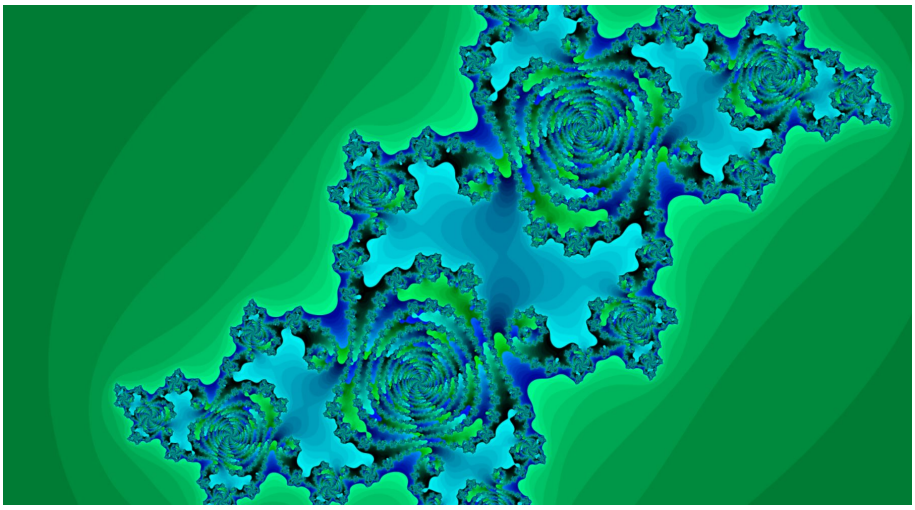


Figure 3.5 Julia set, as obtained from *CUDA Samples*.

3.2 Quick review of MATLAB programming and tools

MATLAB provides a complete environment for high-level programming, targeted to scientific labours. Programming in MATLAB is heavily influenced for other high-level languages, as for example C++ or Java. In order to provide fast programming, a large collection of functions has been built in the software. These functions expand to: linear algebra and equation solving, statistic analysis, file I/O, graph edition, etc.

As seen in Visual Studio or Eclipse, MATLAB IDE also provides a complete editor (Fig. 3.6), featuring similar tools for helping developing process. Debugging tools are also directly implemented into the editor, providing quick access for step-by-step execution, checking variables, etc. Other tools are provided in addition to base MATLAB package, presented as *apps*, expanding its capabilities.

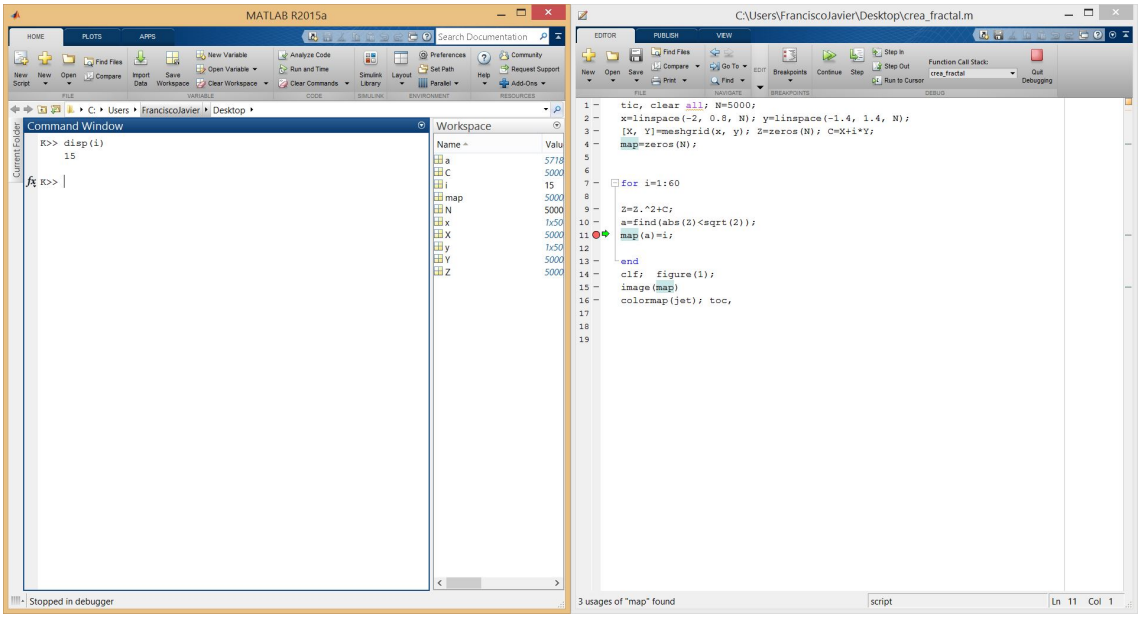


Figure 3.6 MATLAB IDE (left), and editor (right) while debugging a code.

MATLAB programming is considered an interpreted language: code sentences are executed as machine instruction directly without requiring to be transformed into machine instruction prior beginning the execution. This is a common technique for mathematical related programming languages, as Mathematica or R.

This interpretation process is performed by a virtual-machine substrate running on Java, and therefore code performance has been a matter for criticism. In order to speed up execution, several mechanism are implemented on MATLAB: first, **runtime optimization**, known as *Just-In-Time* optimization or *JIT*, provides an efficient way to improve the interpretation of code. A comparison can be seen in [35]. The second one is vectorialization mechanisms, which provides an easy to deploy solutions for vector and matrix operations while reducing the execution time.

In the scope of this work, one tool has been extremely relevant: *MATLAB Profiler*. This tool allows the user to obtain execution time data of code from a simple interface and graphically display of it. Information as time of execution, number of calls, comparison between lines of code, hint to improve code execution, etc. is provided. This features have been represented in Fig. 3.7.

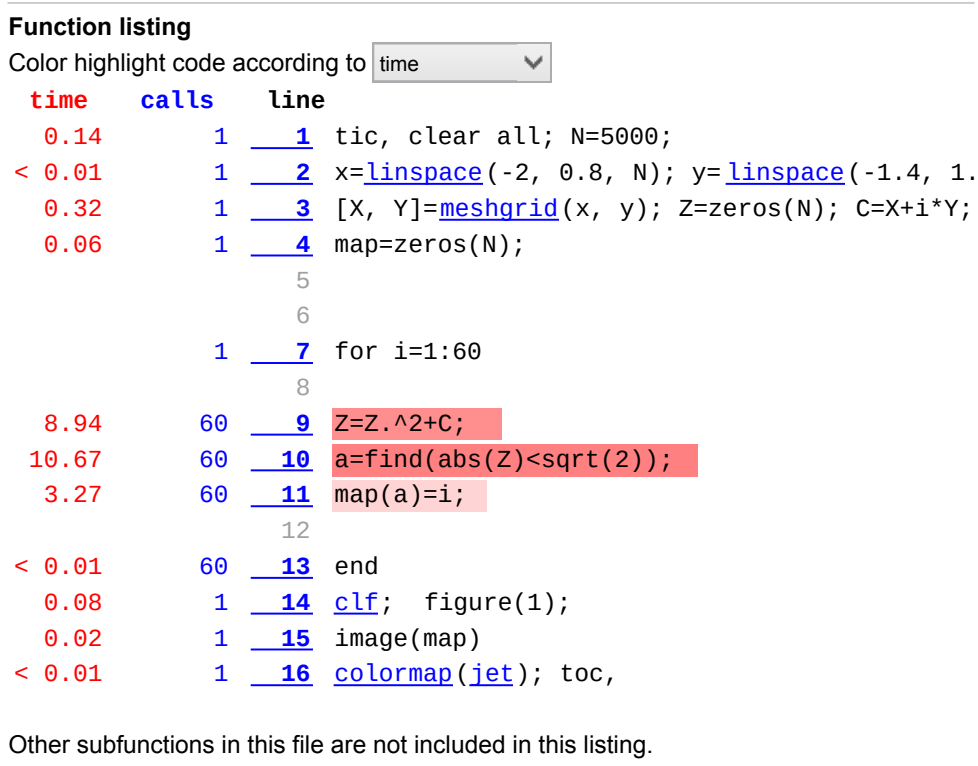


Figure 3.7 Extract of profiling graphical result for a sample code.

3.3 Devices used for code testing

For the development and execution of the algorithms, two different computers have been used, in order to compare the performance over different architectures and environments.

The first testing computer is a Lenovo™ laptop, shown in Fig. 3.8, running Windows 8.1 build 9600 64-bits version. This computer features an *Intel i7-4702MQ* (2.2 GHz) 4-cores 8-threads processor. As a graphic card, it mounts a *NVIDIA GT 740M* CUDA-capable card. An extract of its capabilities is summarized in Table 3.1. Information has been obtained from manufacturers' databases [36] [37] and by execution the card identification tools located at the *CUDA Samples*.

This device has been chosen due to the familiarity with the IDE tools for Windows environment. As being a laptop equipment, special consideration has been taken at the time of running the tests, as power management for portable devices may drastically decrease speed of execution of the solution. This problem has been sorted by ensuring power cord to be connected during the tests, 'High performance' mode to be chosen on 'Power Settings' on Control Panel and by ensuring and ensuring air flow from laptop vents.

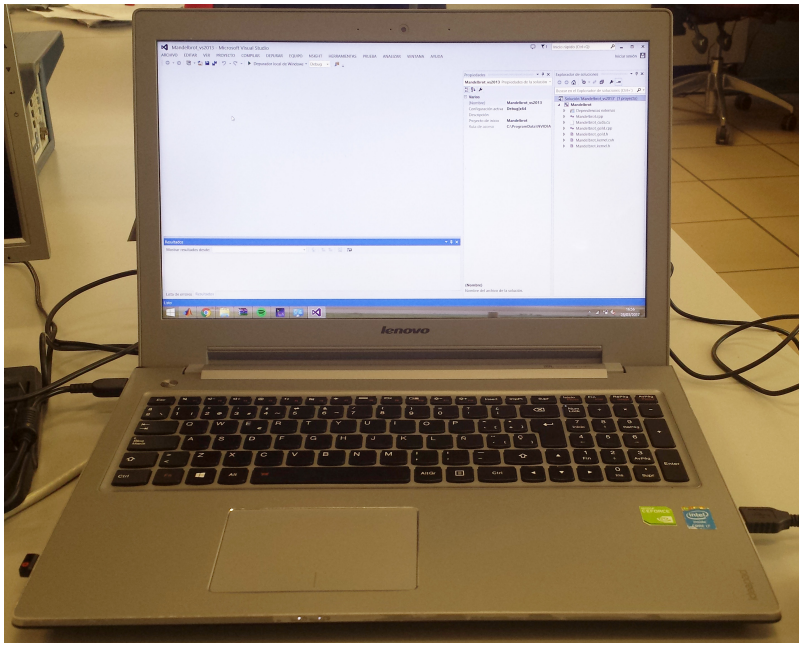


Figure 3.8 Setup of laptop for the tests.

Table 3.1 Laptop-machine specification extract .

CPU	Model	i7-4702MQ
	No. of cores	4
	No. of threads	8
	Nominal frequency	2.20 GHz
GPU	Model	GT 740MQ
	Global memory	2048 MB
	CUDA Cores	384
	Memory bandwidth	14.4 GB/s
	GPU clock rate	900 MHz
	Max. no. of threads per blocks	1024

For higher performance tests, a larger, workstation unit (Fig. 3.9) has been used. This computer runs on Lubuntu operative system, an Ubuntu 16.10 distribution, under Linux kernel 4.8.0. The use of that lightweight Linux distribution over Windows environment may suppose, in addition, a source for potential speedup. This computer mounts an Intel i7-4790K (4.00 GHz) 4-cores, 8-threads CPU. Installed graphic card is *NVIDIA GeForce GTX 780* CUDA-capable.

Figure 3.9 Setup of workstation used for tests.

As having higher capabilities and being a fixed, rack-fitted unit, energy consumption and thermal constrains do not have to be considered on testing process. An extract of system specification [38] [39] can be seen at Table 3.2.

Table 3.2 Workstation specification extract.

CPU	Model	i7-4790K
	No. of cores	4
	No. of threads	8
	Nominal frequency	4.00 GHz
GPU	Model	GTX 760
	Global memory	3020 MB
	CUDA Cores	2304
	Memory bandwidth	288.4 GB/s
	GPU clock rate	3004 MHz
	Max. no. of threads per blocks	2048

For ensuring connectivity to the board during all time of the day, network connectivity has been implemented, allowing access to the device by command line (*SSH*) and by graphical interface, using the tool TeamViewer (Fig. 3.10). Whereas *SSH* provides a fast, lightweight connection with the computer, providing the resources of a system shell terminal, TeamViewer streams host machine desktop to any computer running the guest software and logged in with a previously setup password. The display of a graphical interface allows the designer to take advantage of the available shortcuts and built-in IDE function while easing the compilation and linking process.

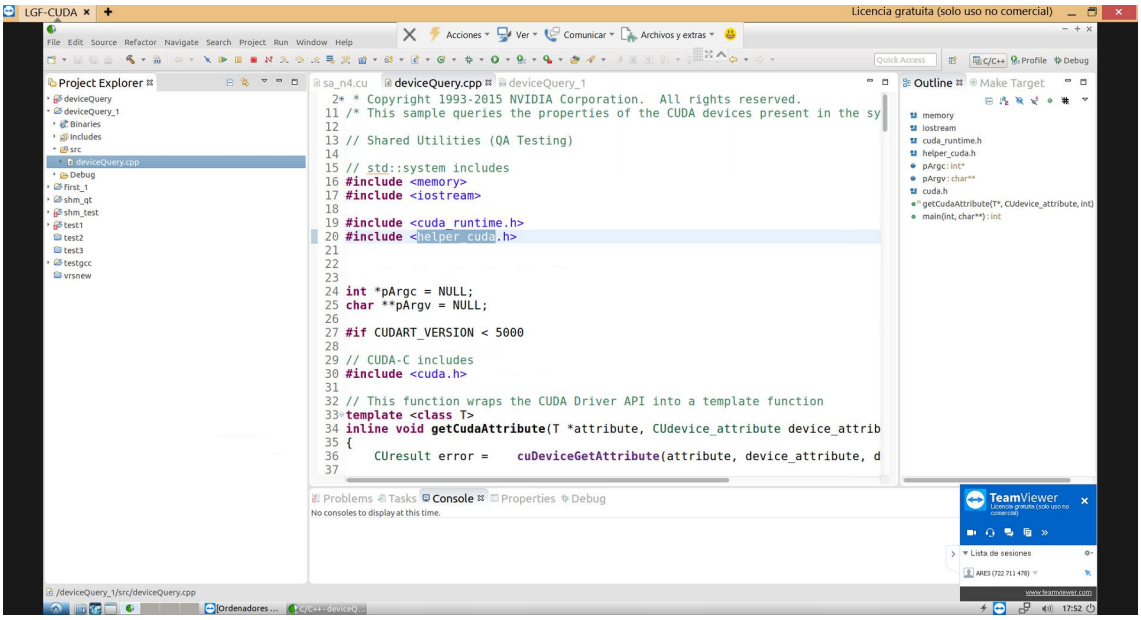


Figure 3.10 Screenshot of connection from guest to host computer, displaying TeamViewer interface and guest desktop and Nsight Eclipse.

4 Simulated Annealing and CUSIMANN

This chapter aims to perform a review of the past work on the implementation of the SHM algorithm for the CUDA environment, particularly on the simulated annealing algorithm. This work was performed by Patricio López under the direction of Ph. D. Ramón Portillo Guisado as a end-of-studies for *Departamento de Ingeniería Electrónica* of *Universidad de Sevilla* for the period of 2011-2012.

The scope of this review can be summarized to two main tasks:

- Perform a *profile* of the present code, discussing the obtained results and proposing ideas for improving its performance.
- Compare the performance of the code for different configuration parameters, as for example compilation code optimization and new compilers.

To illustrate the code, a short description of the SA algorithm as well as its CUDA implementation, CUSIMANN, will be done.

4.1 Description of Simulated Annealing

Simulated annealing was proposed in 1983 by S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi [28]. This algorithm mimics the annealing of metals, a process performed to modify the microstructure of the metals in order increase its ductility and reduce internal stressed. This process consist of heating the metal below the melting point, letting the atoms to reallocate, and progressively cooling it down, recrystallizing on a new, more stable configuration [40].

The simulated annealing considers a potential solution of the problem as the atomic configuration of a metal crystal. The value of the objective function can be directly linked to the energy of the crystal for the present configuration. As the crystal attempts to reduce its energy to reach the most stable condition, the objective function will approximate to a minima.

This process is performed in several iterations, in order to consider the reduction of temperature in the physical system. In the optimization problem, a reduction of the temperature is translated as a reduction in the random component driving the search process.

This process allows the algorithm to start its search from a broad range of values, while the reduction of temperature permits to evolves to better minimum.

The algorithm sequence is presented below:

1. An initial iterand is selected in order to keep the algorithm close to solution. Present solution, $\mathbf{x_0}$ is set to this initial iterand.
2. A value new value $\mathbf{x_1}$, known as neighbourhood solution, is generated in the vicinity of $\mathbf{x_0}$. The lower the simulated temperature, T, is, the closer the neighbourhood solution will be to present solution.
3. The objective function is evaluated for these two values, $f(\mathbf{x_0})$ and $f(\mathbf{x_1})$
   1) A uniformly distributed random value U is generated in range $[0,1]$
   2) If $U < \exp(f(\mathbf{x_0}) - f(\mathbf{x_1}))$, or if $f(\mathbf{x_1}) < f(\mathbf{x_0})$, $f(\mathbf{x_0})$ is set to $f(\mathbf{x_1})$.
4. Jump to **2.** until a fixed number of iterations has been reached. T is not modified.
5. T is reduced a factor of ρ. Unless a *frozen condition* is met ($T < T_0$), jump to **2.**

This program flow is shown in Fig. 4.1

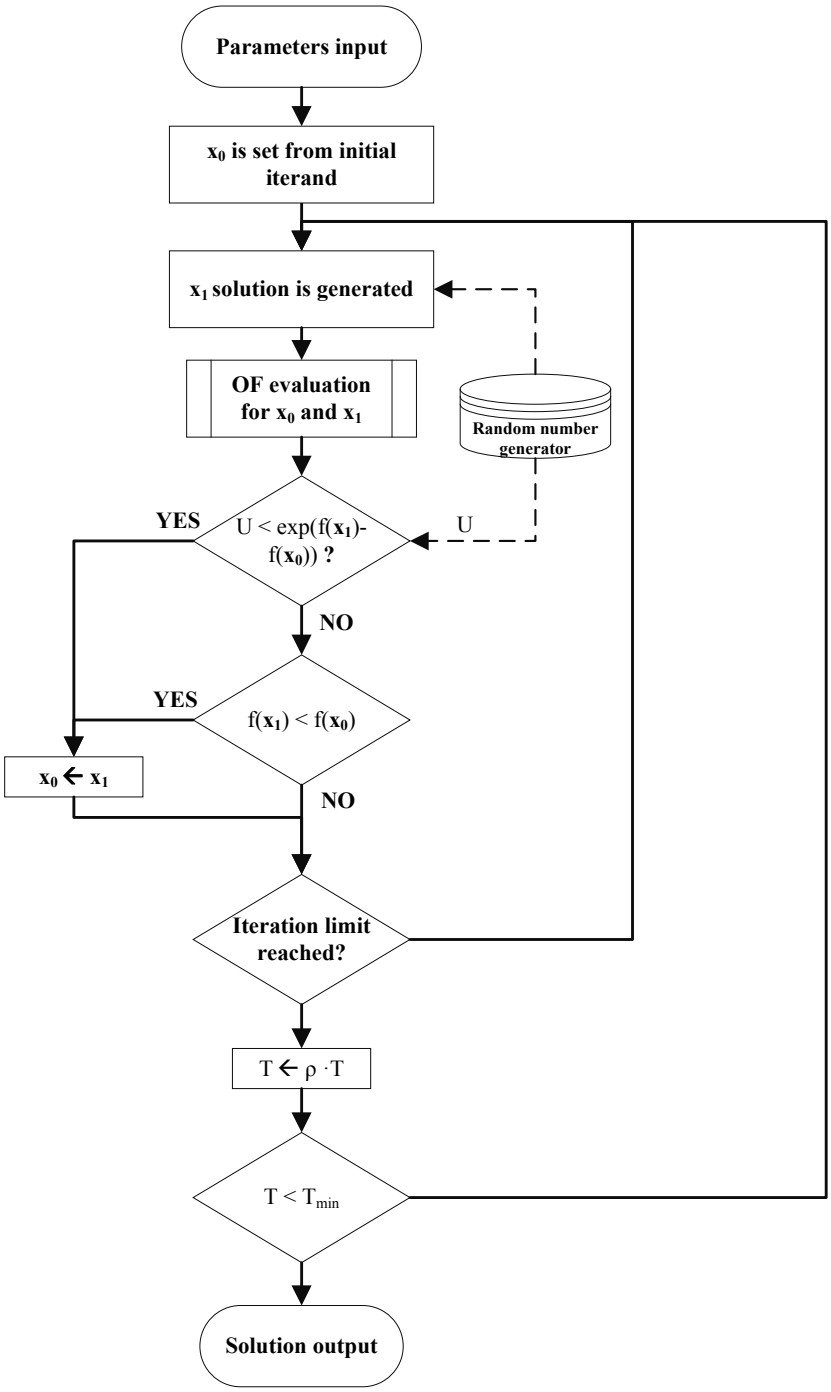


Figure 4.1 SA program flow.

4.2 Parallel SA, CUSIMANN, and SHM implementation

CUDA implementation for simulated annealing algorithm was proposed in 2012 in [41]. This implementation was called **CUSIMANN**.

The underlying idea on this implementation is to consider not only a single neighbour

point, but a larger set of points. These operations can be easily parallelized thanks to CUDA architecture. Therefore, an actual increase in algorithm execution speed can be reach, as evolution achieved per iteration is more significative.

Two groups of tasks are subjected to paralellization:

- Calculation of the objective function, which can be paralellized by assigning each calculation to a single execution thread.
- Selection of the best population, evolution of each population or sorting the populations according to its OF value. Computational resources are divided into several blocks, which address a number of potential solutions. Thanks to *shared memory*, different threads can exchange information, avoiding memory transfers and increasing the performance.

Details of present implementation can be found at [41], as well as open distributed source code.

The implementation of SHM for the CUSIMANN had to deal with two particular issues:

- Detection of compliance for the harmonic content. This was addressed with the addition of a particular *flag* variable obtained by comparing each level with the grid code.
- Solution switching angles have to be applied in ascendant sequence. By using *Thrust* libraries, each population was sorted every iteration.

4.3 Code profiling

By using the Visual Profile provided with CUDA Toolkit, performance figures have been obtained. This information will help to characterize the execution of the code and which sequences of it suppose the longest calculation time.

First performance figure to obtain is the program timeline. This has been represented in Fig. 4.2, and a detailed view of it can be seen in Fig. 4.3. This graphs have been obtained for a sample execution at $m_a = 0.9$, for the 15 switching angles , 17 harmonics into mitigation SHM problem, under the grid code in Table 1.1.

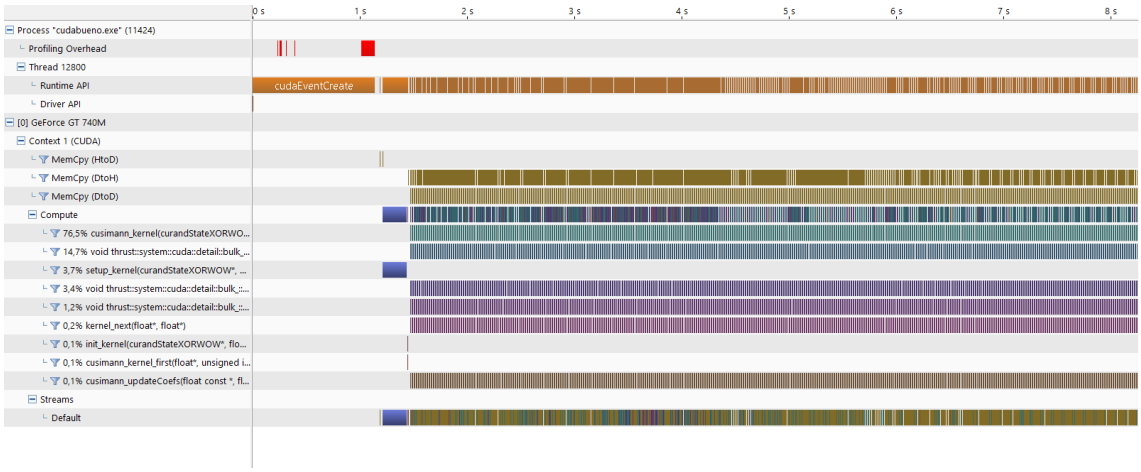


Figure 4.2 Complete execution timeline.

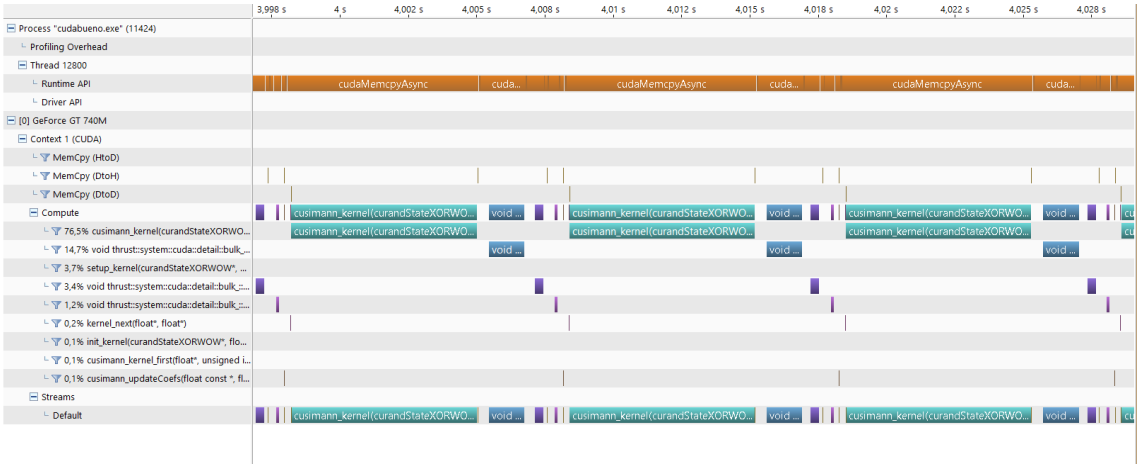


Figure 4.3 Detailed execution timeline.

Information provided in Figs 4.2 and 4.3 has been processed and a comparative of time of execution for the high-level functions is presented in Fig. 4.4.

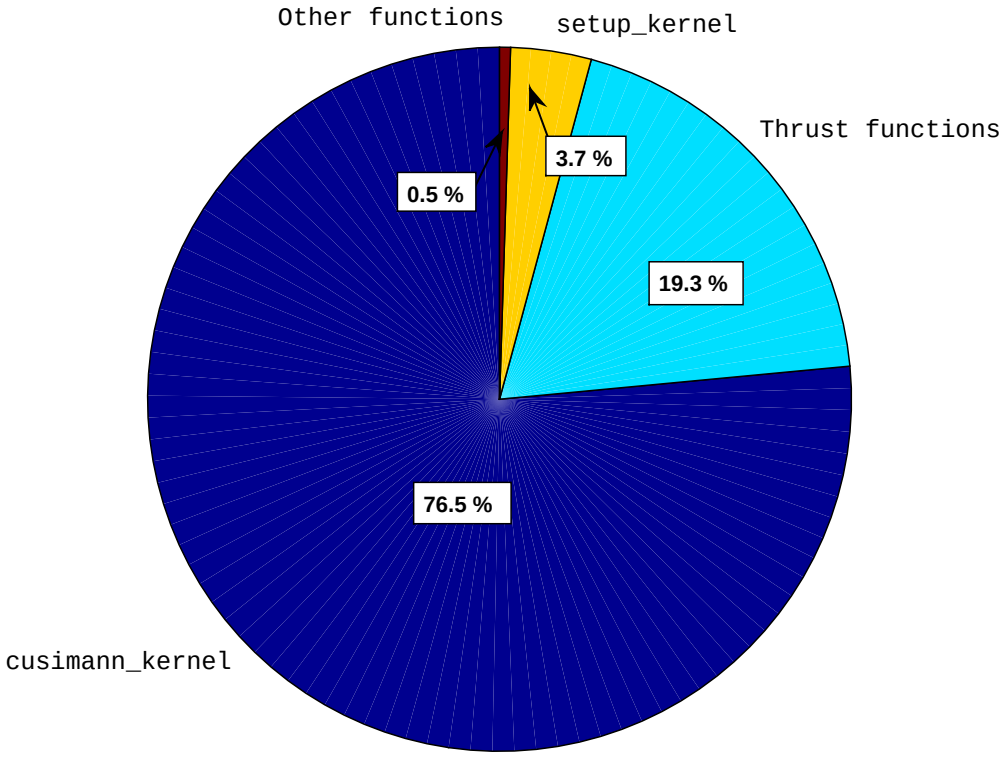


Figure 4.4 Distribution of time for high-level functions.

As it can be seen in Fig. 4.4, `cusimann_kernel`, which calculates the neighbourhood points and evaluates the objective function, supposes the longest part of execution time. As objective function is a device subfunction, no particular data can be obtained in order to benchmark its performance. It is proposed a review of the code involved in this function, in order to find sources of lost of synchronization or bottlenecks increasing the execution time.

In addition, Thrust sorting functions also accounts for a significative time of execution. It is proposed future implementation of the sorting function by manually computing this process. Even though Thrust ensures good performance for many situation, it is possible that manually optimized function to fit better the requirements of the problem.

It is also relevant to consider the time requested by cudaEventCreate processes, as seen in Fig. 4.2. This function is required for execution measurements, and therefore have to executed on profiling tests.

Another information provided from the profiler areas where the code can be potentially improved. For the algorithm, all of this suggestions are related to memory management:

- *Low Memcpy/Compute Overlap*: Transfer of information between host and device are performed sequentially, even though they could be performed in parallel with algorithm evolution.
- *Inefficient Memcpy Size*: Copy of data from and to host are performed in size below the recommended.

- *Low Memcpy Throughput*: Similarly as above, bandwidth for copy operation is only used in a 1.6 % of its maximal capacity.

It is also possible to obtain a rank of functions to be optimized. The higher the rank, the more the overall performance can be optimized. The results are presented in Table 4.1.

Table 4.1 Optimization priorities ranking.

Function	Points obtained
`cusimann_kernel`	100
`Thrust functions`	20
`setup_kernel`	5
`cusimann_updateCoefs`	1

As represented, `cusimann_kernel` function has to be devoted higher optimization efforts, in order to achieve greater improvement in global performance.

Resource occupancy and availability is also analyzed on Profiler. It is possible to compare the computing and memory resources used on the execution on the program, in contrast to the limits available in the host computer. It is proposed to modify resource dispatching in algorithm configuration in order to approximate the resources to higher bounds.

Results from this process are presented in Fig. 4.5. As it can be seen, arithmetic operations accounts for the higher percentage on resources being used. However, unused resources are above the 50% of total, which evidences room available for further improvement. As described before, memory results sum up to a small percentage of available resources. In order to keep memory management on efficiency, it is desirable to process as less data as possible and transfer it at the highest volume as possible.

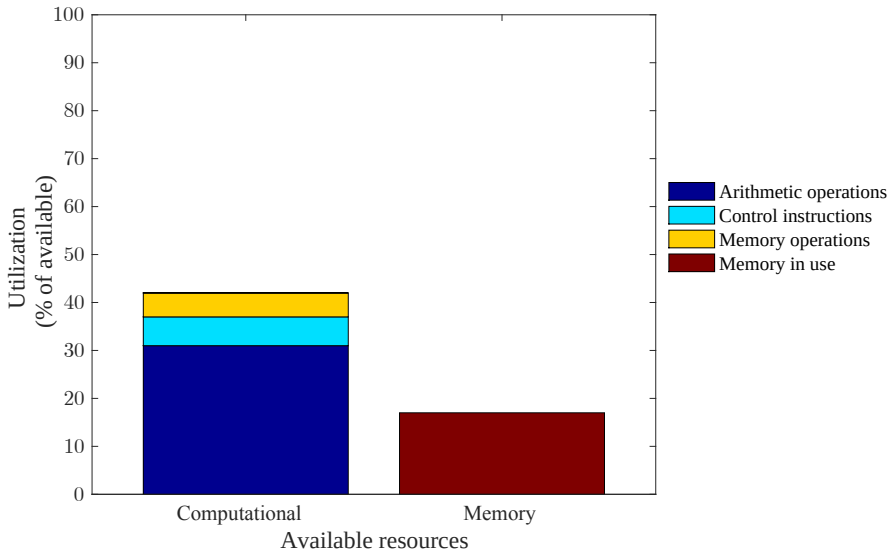


Figure 4.5 Utilization of computing and memory resources.

The last relevant information presented in the Profiler is the occupancy report. This report analyze the CUDA resources -blocks, warps and threads- being used. The results are presented in Table 4.2

Table 4.2 Occupancy achieved by the algorithm.

Variable	Achieved	Device Limit
Active blocks	16	16
Active warps	32	64
Active threads	1024	2048

The information presented shows the available room for optimization, in order to increase the distribution in terms of warps and threads per block.

4.4 Proposed changes and benchmarking

This chapter aims to compare the execution speed of the algorithm under the new compilation environments available.

First parameter to consider is the **CUDA Compute Capability**. In order to take advantage of new architecture and functionalities available in more recent modern cards, CUDA Compiler is implemented with support for these features. Nowadays, CUDA Compute Capability version is 3.5 for version 8.0 of its toolkit.

In order to select a certain capabilities for compilation, the `sm_XX` is included into the compiler toolchain. For our study, versions 1.3, 2.0, 3.0 and 3.5 will be tested.

In addition, CUDA Compiler allows to perform an optimization process during the compilation of the program. Three optimization targets are implemented into the compiler: *Minimize size* (`O1`), *Maximize speed* (`O2`) and *Full optimization* (`OX`).

Results for execution of the algorithm are presented in Table 4.3. The results have been obtained for the 15 switching angles, 17 harmonic mitigation problem, for a total of 5 executions every value.

Table 4.3 Execution time (in seconds) for different *Compute Capability* and optimization target parameters.

		Optimization target			
		No optimization	**Minimize size**	**Maximize speed**	**Full optimization**
Computation Capability	1.3	5.713	5.708	5.698	5.683
	2.0	4.132	4.199	4.106	4.123
	3.0	4.179	4.194	4.115	4.150
	3.5	4.272	4.225	4.224	4.245

In addition, in order to provide a comparative figure, (Table 4.4) presents the reduction in execution time achieved in comparison to the slowest data -1.3 version with no optimization.

Table 4.4 Percentual reduction in execution time (respect to 1.3 No optimization) for different *Compute Capability* and optimization target parameters.

		Optimization target			
		No optimization	**Minimize size**	**Maximize speed**	**Full optimization**
Computation Capability	1.3	n.a	0.087	0.26	0.53
	2.0	27.67	26.50	28.13	27.36
	3.0	26.85	26.59	27.97	27.35
	3.5	25.22	26.05	26.06	25.70

As it can be seen, a step change is observed from performance in 1.3 version to 2.0, which can be associated to newer board capabilities to be used. However, no significative change is observed between 2.0, 3.0 and 3.5. This can be related to board architecture, as newer capabilities are targeted to increase the performance of newly released boards.

In order to obtain the difference between each optimization level, percentual change in change over no optimization for every version are presented in Table 4.5.

Table 4.5 Perceptual reduction in execution time (respect to 1.3 No optimization) for different *Compute Capability* and optimization target parameters.

		Optimization target		
		Minimize size	**Maximize speed**	**Full optimization**
Computation Capability	1.3	-0.087	-0.26	-0.53
	2.0	+1.62	-0.63	-0.22
	3.0	+0.36	-1.53	-0.69
	3.5	-1.10	-1.12	-0.63

It is interesting to notice that best configuration for execution time are *Maximize speed* and **Full optimization**, while *Minimize size* can increase execution for several versions. Greater optimization figure is -1.53%, as an evidence that further optimization has been done by implementing significative changes in code structure.

5 Exchange Market Algorithm application for SHM problem

The following chapter aims to introduce a different heuristic algorithm for solving the SHM problem, the Exchange Market Algorithm. This new algorithm, proposed on 2012 by Naser Ghorbani and Ebrahim Babaei in [29], treats the nonlinear optimization problem as shareholders behaving in the stock market. This algorithm has proven good performance in terms of number of iterations and convergence in comparison with advanced methods as GA, PSO, etc [42]. By implementing the SHM cost function and its restrictions, our target is to reach a solution decreasing execution time, which also involves other algorithms for code reliability and convergence.

The scope of the present chapter would be introducing the principles of the algorithm and its application for the SHM problem, covering the following issues:

- Technical description of the algorithm, its parallelism with the stock problem and achieved results by the autors.
- Implementation of a basic structure for solving the SHM problem, with no direct control on iteration stop.
- Profilling of the code, in order to obtain a performance figure of the several blocks involved in the program.
- Implementation of an advanced structure, in order to solve the issues of initial iterand and execution termination.
- Design of a basic interface, in order to perform calculation without the necessity of directly manipulating the code

5.1 Description of the algorithm

As described above, the EMA takes its inspiration on shareholders objectives when operating in a competitive stock market. In a real world, price of stocks are affected by a vast number of parameters, but being mainly correlated to the demand. Every shareholder will tend to maximize its benefit by exchanging stocks or holding them. The expected behavior of each

shareholder is affected by its market position -higher ranked shareholders will assume less risks than those on a lower rank- and how oscillating is the market in question.

The authors stablish a parallelism between the different potential solution for the optimization problem and the different shareholders operating in the market. Every shareholder has a unique portfolio of stocks, every of them with a value on the market, and they sum up to a total benefits or losses. These benefits are directly correlated with the evaluation on the objective function on this solution.

In real market, every shareholder behaves in a competitive way against the rest, in order to reach the maximum benefit [43]. This iterative process is based on the experience of each operator and relies past successes and failures. The expected behaviour of each shareholder is affected by its market position: higher ranked shareholders will assume less risks than those on a lower rank, which will attempt riskier practices.

Moreover, shareholders can trade their stocks in different markets, which can be classified in terms of its stability. Stable markets tend to maintain stocks prices within minor fluctuations, however leaving less room for rapid increase on benefits. Oscillating markets, in contrast, allow rapid changes in the value of the stocks, while increasing the risks of shareholders. Market stability is generally defined by political and governmental factors, as well as historical development of the market in question [29].

This philosophy was interpreted by the authors as a reliable strategy on the field of nonlinear optimization, therefore expanding the concept of two different markets to the process of combination -trading, sharing- among the potential solutions to the problem. This way, the two following modes where proposed:

- **Non-oscillating mode**, which correlates with balanced or stable markets. Stocks prices are less likely to change, so that shareholders will tend to *change* their actions in order to increase their benefits slightly. In the algorithm implementation, a *change process* will be established between two of the solutions of the population, and will suppose a minor change on their fitness to cost function.
- **Oscillating mode**, which directly relates to oscillating market. The market price is subject to a higher price variability, so that the shareholders will follow a *trading* process, trying to extract the highest benefit from the variable market conditions. On the implementation, this condition is mimicked by the addition of a random component to each solution participating in this process. As in a real market, only worse ranked shareholders would trade their stocks, not all the solutions are subject would in the *trading* process. Those solutions which are worse conditioned will account for large random components than better ranked ones.

As introduced before, the algorithm deepens into the different behaviours of the shareholders, as their decisions may be more conservative or riskier depending on their position on the market. On the proposed market, three different groups can be selected:

- *High rank shareholders*, (10-30% of total population), which does not expect any changes on their stocks and will not take any risk.
- *Mean rank shareholders*, (20-50% of total population), which keep on the target for best shares among all the available ones, trying to stand apart from the mass.
- *Weak rank shareholders*, which stand with lower values on the objective function, will follow riskier actions than mean-ranked to increase their rank.

In order to translate the several ideas into an algorithm, the ranking, trading and sharing processes have to be thoroughly characterized, in order to obtain implementable equations. Reader is invited to proceed to [29] in order to obtain detailed information on how this conditions have been translated into algebraic equations. Randomness affecting both of these processes and the thresholds for grouping the solution can be configured, in order to increase the compatibility of the problem with the algorithm.

By including several control flow structures, as for example tracking the number of iterations completed and comparing to a bound, the algorithm is ready for being implemented in a programming language

A descriptive pseudocode of the process is show below:

1. An initial population is generated, which will act as the shareholders.

2. While a termination criteria is not met,

1. The value of the objective function is calculated for all the potential solutions on the population.
2. According to the values obtained in 1., different solutions are ranked in terms on fitness and groups as established according to the defined thresholds.
3. Balanced market condition is considered:
   1) Mean-ranked *shareholders* (solutions) apply a *changing* process to their shares in non-oscillating mode
   2) Weak-ranked *shareholders* (solutions) apply a *changing* process to their shares in non-oscillating mode.
4. New values of objective function are evaluated, ranked and groups re-established, as done in 1. and 2.
5. Oscillating market condition is considered:
   1) Mean-ranked *shareholders* apply a *trading* process to their shares in oscillating mode.
   2) Weak-ranked *shareholders* apply a *trading* process to their shares in oscillating mode.
6. Return to **2.**.

3. The best solution has been found within the bounds of the criteria.

This program flow has been represented as a flow diagram in Fig. 5.1

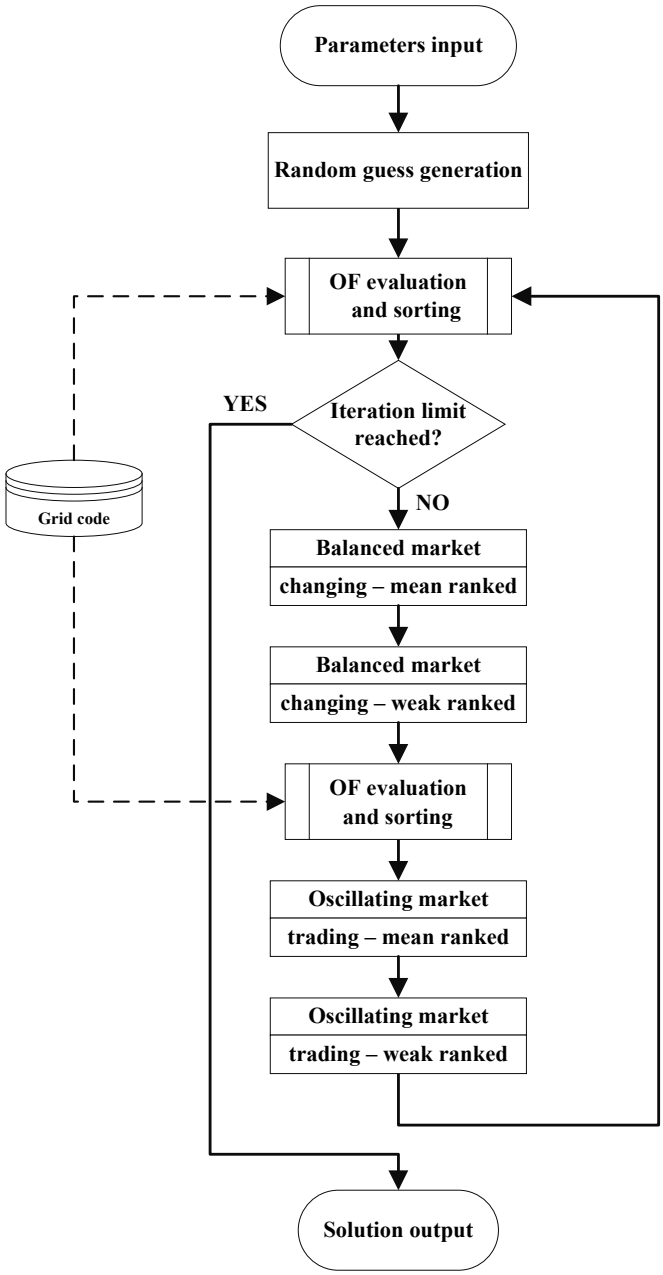


Figure 5.1 Code flow diagram.

EMA has shown so far better results on comparison with other typical algorithms in terms of convergence for a certain number of iterations. This improvement has been estimated by the authors in several order of magnitudes for representative functions in the field of optimization benchmarking, as for example Schwefel function (eq. 5.1) or Ackley function (eq. 5.2) [42] [44].

$$f(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{5.1}$$

$$f(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{5.2}$$

An extract of the performance obtained and presented by the author in [29] is presented in order to illustrate EMA performace again methods as GSA and PSO. EMA shows an important improvement in performance in comparison to other well-positioned algorithm in the state-of-art. The results are shown in Table 5.1.

Table 5.1 Time of calculation for EMA, GSA and PSO for Acker and Schwefel function, in seconds.

Function	EMA	GSA	PSO
Acker	32.13	91.12	41.60
Schwefel	31.85	95.77	38.86

5.2 MATLAB implementation

The implementation has been performed in MATLAB software, due to its capabilities for handling long matrix data and optimized vector operations. Low-level software implementation can be executed in future review of the code, in order to take advantage of the available complete hardware capabilities.

For the implementation of the SHM, the optimized code from the authors has been modified in order to adapt the objective function of the problem. This objective function is due to consider a certain number of harmonics, including the main harmonic, and addressing also the value of the THD for those harmonics. By implementation of a suitable model, more complex considerations can be studied, as for example economic aspects, as output filter price, which is closely related to the harmonic content presented on the output waveform of the inverter.

The code in consideration is divided in several modules, corresponding each one to a concrete task in the execution. These modules are listed below:

- `EMA.m`, which provides the base function for execution of the algorithm and for the user to set the parameters (m_a parsing) and displaying the results (best solution, value of OF, etc.).
- `Initial.m`, which includes algorithm parameter for the management of the populations. Function specific parameters -proportion of solutions in each group, weights of the risks taken by each group, etc. are defined in this module.
- `SHM.m`, which includes the configuration of the SHM in question, i.e. grid codes and number of harmonics which will be addressed and their indexes.
- `oscillation.m`, which executes the algorithm operations in oscillating mode.
- `notoscillation.m`, which executes the algorithm operations in not-oscillating (balanced) mode.

- `fitnessN.m`, which provides the value of the objective function for the desired angles array.

A graphical representation of these structure is presented in Fig. 5.2

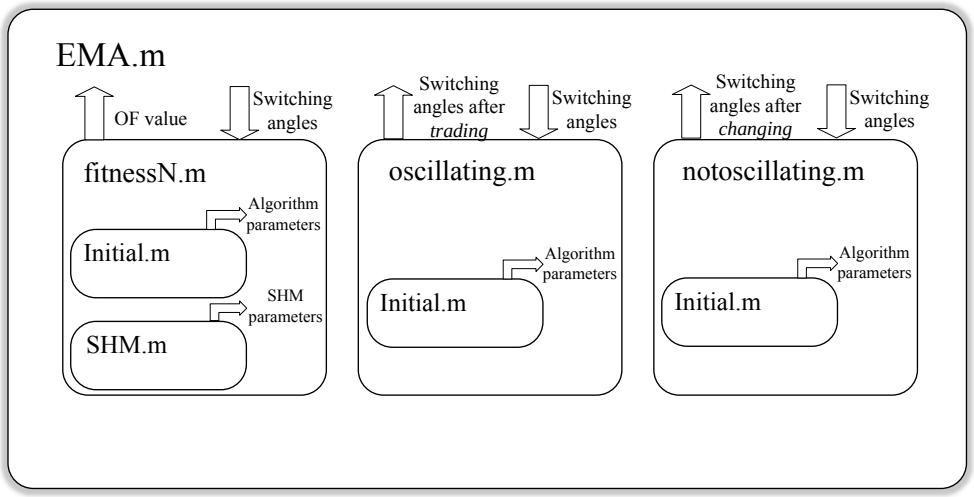


Figure 5.2 Function hierarchy.

As discussed in the chapter 1, the main drawback in terms of computation cost of the SHM is the complexity of the cost function, so that special care has been devoted on its implementation. Moreover, the cost function has to be evaluated twice on each iteration for each element in the population. As a result, a reduction of several milliseconds per execution of the OF may suppose reductions of seconds in total time.

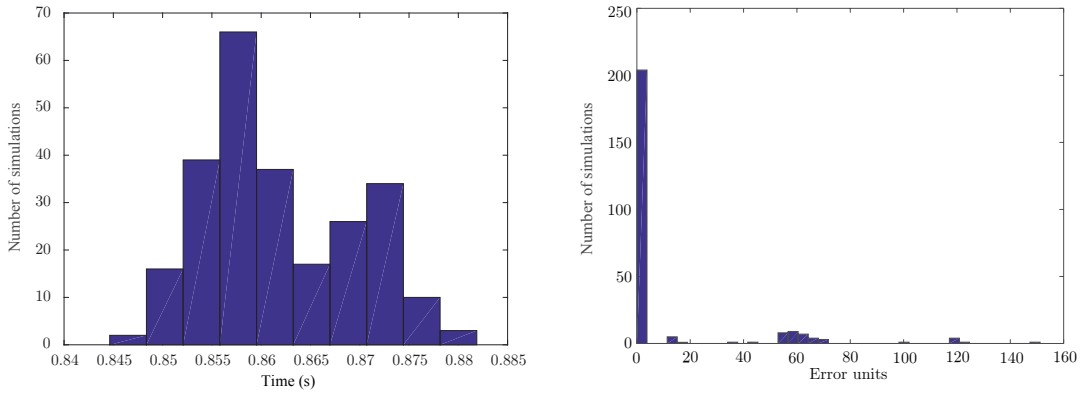
For the following tests, unless said so, the preferred configuration for the EMA algorithm is: 15 switching angles, mitigation up to 49^{th} harmonic and maximum harmonic distortion limits imposed by EN 50160 + CIGRE WG 36-05 grid codes (Table 1.1). Results provided in this document are mainly focused for the m_a range of 0.75 to 1.2, due to availability of initial iterands suitable for the range.

5.3 Results obtained on initial implementation on MATLAB

In a first approach, a batch of tests has been conducted on the laptop machine, for addressing the most adequate configuration of the algorithm. In these tests, a limit on the number of iterations will be set, and the output results will be compared in time of execution and error on the objective function. In the first approach, no additional algorithms for execution control will be used and a random initial population will be used for each simulation. The problem in study will calculate 6 switching angles for addressing 7 harmonics, including the fundamental harmonic, and also the value of THD, in compliance to the grid code. In order to ensure the reliability of the algorithm, each simulation is repeated for a relatively large number of times, so that a significant values are found and a significant distribution of data is obtained.

The accumulated values for this batch of tests is shown in Fig. 5.3. Code execution have been limited for 500 iterations. Data set is composed of 250 executions of the algorithm for

a $m_a = 1.1$. Fig. 5.3(**a**) features the distribution for the time of execution, while Fig. 5.3(**b**) features the distribution for the value of OF for the solution obtained by the algorithm



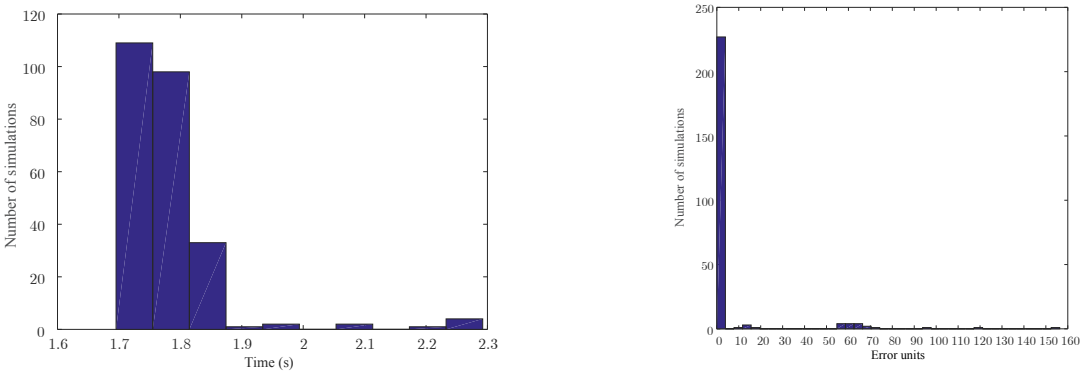
(a) Time of execution.

(b) Best value of OF..

Figure 5.3 EMA performance for a total of 500 iterations, $m_a = 1.1$.

As in 5.3, an important number of simulations, nearly 20%, are shown as not converged to a valid value according to the grid code as evidenced by the larger values of OF -percentile 80 is found at 0.523 error units. Therefore, it should be considered to increase the assigned number of iteration to fulfil with grid code standard. In contrast, a percentile 90 of time of execution can be located at 0.874 seconds. Values appear to be distributed closely to the mean value of time, which can be deduced by the highly correlation between time and number of executions.

In order to obtain a reduction on the number of non-converged solutions, a more exhaustive approach can be followed by increasing the number of iteration to 1000. This new test will be performed under the same consideration as described previously. Better convergence, under the cost of longer times of calculation can be expected. Results for those calculations are presented in Fig. 5.5

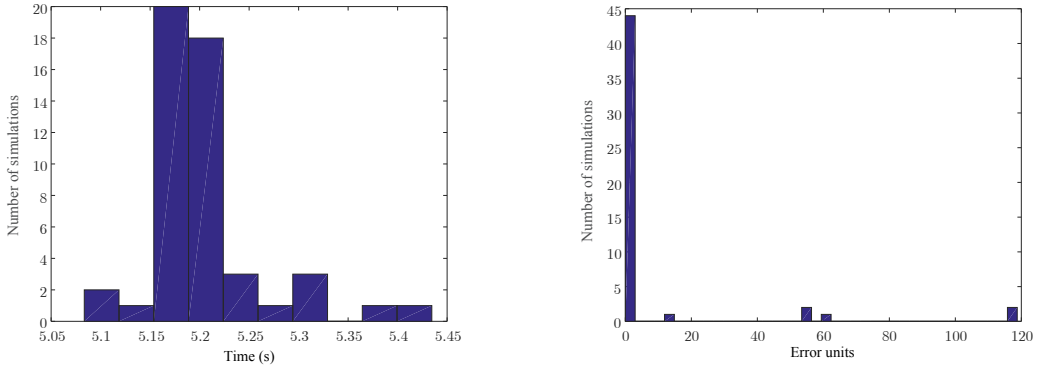


(a) Time of execution. **(b)** Best value of OF..

Figure 5.4 EMA performance for a total of 1000 iterations, $m_a = 1.1$.

In this new situation, Fig 5.4(**b**) shows a larger number of converged simulations -percentile 90 for OF located at 0.759-, which confirms the value of 1000 iterations as being more closer to the required number of iterations for the present configuration. However, the mean time of calculation, as seen in Fig 5.4(**a**) has therefore risen up -percentile 90 at 1.836 seconds, nearly doubling the time required for half the number of iterations.

In order to fully characterize the behaviour of the algorithm, a last test will be performed by increasing the number of iterations to a much higher value, as for example to 3000 iterations. For this case, only 50 simulations have been taken into account. Results are presented in Fig. 5.5.



(a) Time of execution. **(b)** Best value of OF..

Figure 5.5 EMA performance for a total of 3000 iterations.

It can be clearly seen that the execution time has risen up to much higer values -percentile 90 at 5.29-, as seen in Fig 5.5(**a**), while the value of OF shows mostly all the calculation as converged -percentile 90 at 0.438, Fig 5.5(**b**). This test proves that, in order to reach a trade-off between execution time and convergence, new methods for controlling program flow, as for example detecting compliance of each harmonic to the grid code, have to be implemented. By using that, on each calculation the algorithm will only execute for as much time as required, and thus using the computation capabilities efficiently.

In order to obtain a characterization on how the algorithm evolves on every iteration step, OF values can be sampled on every iteration. Due to the formulation of the EMA, on each iteration OF will be equal or lower than calculated for the iteration before. The target for the SHM problem is reducing the value of OF until all the harmonics have converged to a compliant value according to the grid code. The value of OF, according to the function definition in 1.5 would be closer to zero when convergence has been reached.

By representing all these values, evolution of the algorithm can be obtained, as shown in 5.6.

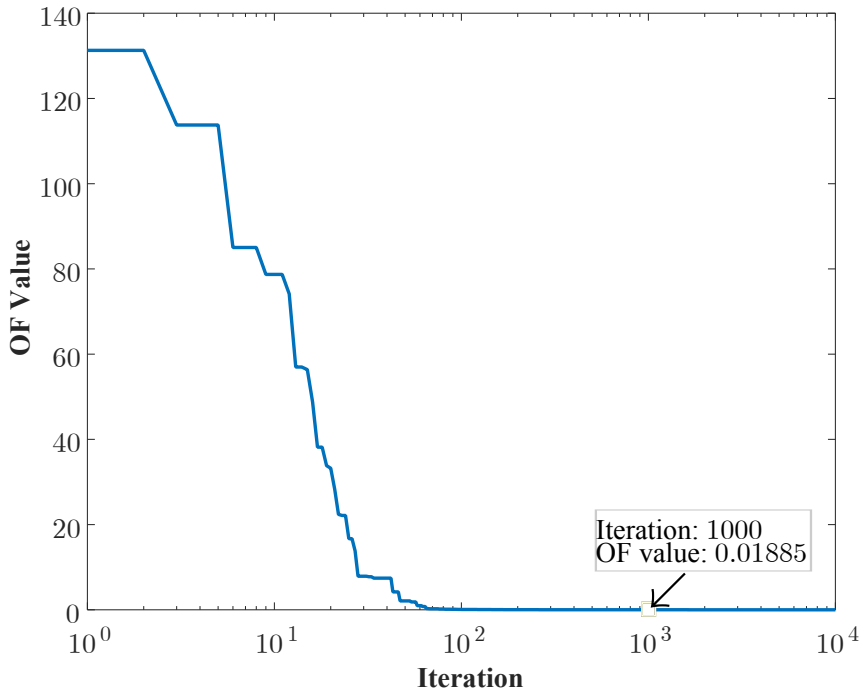


Figure 5.6 Value of OF as the algorithm evolves, $m_a = 1.1$.

As it can be seen, EMA is capable of steeply reducing the OF value on the first hundreds of iterations, as an important number of harmonics reach convergence on this period. However, a significative number of iteration is required on the last iterations to reach compliance of few high order harmonics.

To summarize the results obtained in this first batch of tests, we can extract the following ideas:

- The EMA provides a fast convergence in terms of number of iterations.
- The number of iterations is sensitive to the initial iterand, so that a mechanism for providing accurate initial iterands must be developed.
- In many situations, the algorithm has provided a solution within the grid code in a smaller number of iterations than the upper limit. For reducing the executing time, program flow can be ceased after the valid solution is met.

Next sections will focus on knowing how much each component of the implementation is contributing for the total execution time, which is commonly known as *profiling* and implementing the ideas suggested in this first implementation.

5.4 Code profiling

As a mean to find bottle necks and better understanding on how computing resources are being distributed among the different functions composing the algorithm, MATLAB profiling tools are proposed to perform this analysis. Code profiling provides information about time of execution of particular code instructions and number of times a code line is being called. This tool also compares each functions against others, providing visual reference for worst structures in terms of execution time. Code profiling is nowadays a very relevant debugging tool, as can provide the programmer hints of possible unexpected behaviours.

An important consideration on profiling tools is that execution speed may be significantly slower than the achieved by normal execution of code. This can be tracked to the suppresion of *just-in-time* optimization mechanisms, as implemented in MATLAB, so that reference time must be compared against time reported on normal execution.

The present profiling has been obtained by execution of the algorithm on the SHM problem for 6 switching angles for addressing 7 harmonics, similarly as performed in Figs. 5.3-5.5, and for a $m_a = 1.1$. This execution have been conducted on the laptop machine, with a limitation of 1000 iterations.

The results provided for the high-level functions composing the EMA can be seen in Table 5.2. Three magnitudes are represented in this table:

- Number of calls of every function considered
- *total time* this function has been running
- *self time*, which considers the time each function has been in the execution foregroud

It is important to make the difference between these two last magnitudes. A function execution process is summarly defined as: **1)** the function is called, **2)** function is on foreground, and its commands are being executed, **3)** the function calls to a subfunction, the instruction pointer is stored in the stack, so function is no longer in the foreground, **4)** after the subfunction is completed, the instruction pointer is recovered from stack and original function is on foreground again, and **5)** function execution is completed.

These terms can be used to explain the difference between total and self time: total time considers how long the function has been running, i.e. time since the function has been called until it is halted. This time include the time it has been elapsed on executing subfunctions. In contrast, self time only considers the time the function has been on the foreground, i.e. executing commands contained on the body of this function and not in subfunctions defined on it. A further explanation can be found at [45].

Table 5.2 EMA Profile summary for $m_a = 1.1$, 6 switching angles and 7 harmonics.

Function Name	Calls	Total Time	Self Time
`EMA`	1	6.829 s	0.093 s
`fitnessN`	1999	5.414 s	5.393 s
`oscillation`	999	0.997 s	0.997 s
`notoscillation`	999	0.320 s	0.316 s
`Initial`	3998	0.017 s	0.017 s
`SHM`	1999	0.008 s	0.008 s

Results in Table 5.2 shows that OF evaluation `-fitnessN` accounts for the higher time of calculation, as it can be consider for large number of evaluation of non-linear operators -mainly *sin()*. Evolution functions, as `oscillation` and `notoscillation` also accounts for an important amount of time, but smaller as operations involved are mainly arithmetic -especially for `notoscillation`- and generation of random numbers -especially for `oscillation`. Other functions accounts for a neglectable time of execution `-Initial` and `SHM`- as they are mainly for setting initial configuration to other functions.

A graphical representation of Table 5.2 has been performed in form of pie chart, as shown in Fig. 5.7

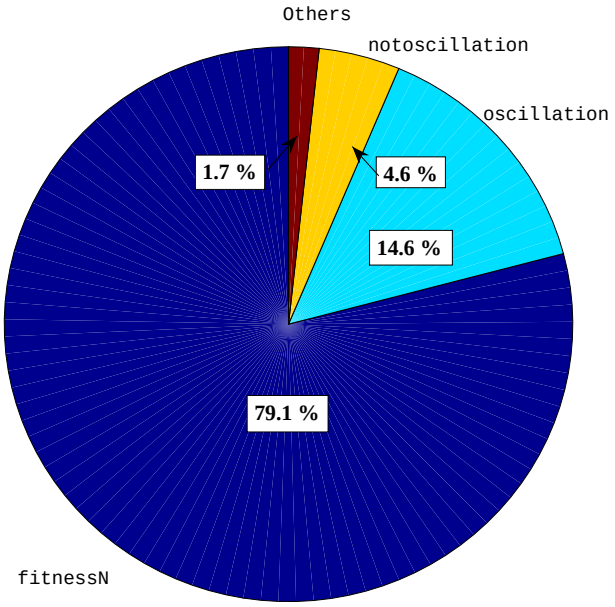


Figure 5.7 Comparison of function execution time for EMA .

Profiler can also provide an in-look inside the execution time of a certain function, in order to know how execution time is distributed among the different commands that take part in the execution. This tool can provide useful information for improving the code. Table 5.3 shows results obtained for profiling on fitnessN function.

Table 5.3 `fitnessN` profiling results .

Line of code	Total Time	% Time
`Ej=Ej+sin(tabla(num_har)*pop(k,:))*tresb`	1.536 s	31.0%
`FC(k,1)=FC(k,1)+cj*Ej.^2`	0.633 s	12.8%
`pop(k,:)=sort(pop(k,:))`	0.340 s	6.9 %
All other lines	2.442 s	49.3%

Similarly, a graphical representation to the results obtained in Table 5.3 is provided in Fig. 5.8.

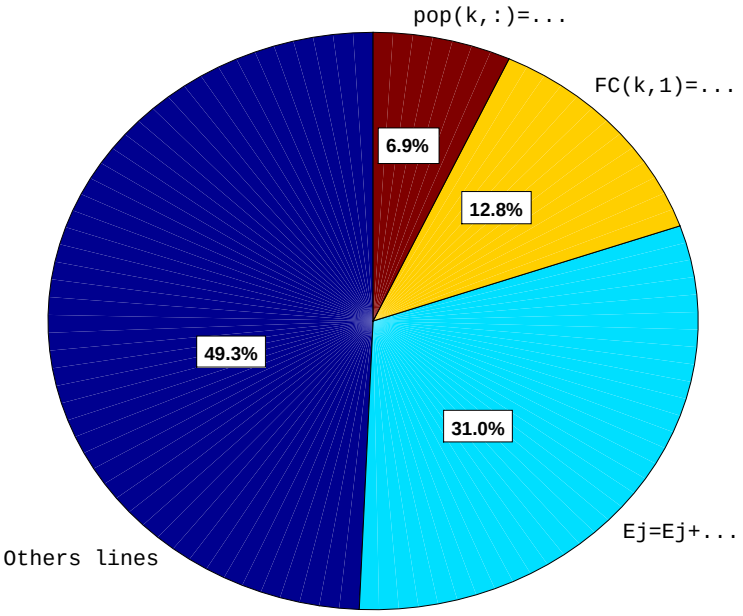


Figure 5.8 Comparison of function execution time for EMA.

As it can be seen, the evaluation of harmonic component and subsequently squaring and adding to objective function are by far the most consuming tasks in execution. This is a proof on how important is adapting the code to the programming paradigm under consideration. MATLAB, as said before, is known for its vectorialization capabilities, i.e. expressions can be written in terms of matrix and arrays, which provides a faster and more reliable implementation than loop approach that can be found in C. While this improvement cannot be sensed in small programs, when the data set, as here presented, grows up, vectorialization provides always the better results.

5.5 Implementing advanced features for EMA

Following the proposed ideas from previous batch of tests and analysis, this section will aim to develop and test the code performance after implementing a solution to those ideas.

First topic to address is the issue of code initialization, as previous tests proved a stiff dependence on the value elected in order to start the algorithm calculation. In order to provide the algorithm with initial iterand close to the final compliant solution, a database of initial iterands have been implemented. These iterands are defined so that the solution for a certain range of m_a is close to the value provided, obtained by gradually exploring the solutions obtained as m_a is slowly changed. By selection of those initial iterands reduces the initial value of OF and, therefore, the required number of iterations. As an example, for values of $m_a = [0.2, 0.67]$ the solution [0.017609, 0.099066, 0.133830, 0.334835, 0.360686, 0.452336, 0.474480, 0.569505, 0.587350, 0.685097, 0.698647, 0.793646, 0.832639, 1.263794, 1.293530] is a good iterand for the SHM problem with 15 switching angles and 17 harmonics. This data has been obtained from previous works on the SHM algorithm, as performed by Patricio López under the direction of Ph.D. Ramón Portillo Guisado, or in [20].

To accomplish this task, a library of initial iterands has been implemented, `initial_pop` , which compares the definition of the problem -number of switching angles and modulation index- and returns a solution within a reference interval of the final solution.

In order to efficiently stop the execution time when a compliant solution has been found, a code termination condition has been proposed. For this issue, the program flow defined in **Section 5.1** has been modified, in order to provide a faster termination in case of detecting that the best solution complies with the grid code limits. The following modification on the program flow is proposed bellow:

1. i^{th} iteration is executed.
2. The OF evaluation function, `fitnessP`, provides the value of OF and a flag for each solution indicating if compliance with the grid code is satisfied in all the harmonics into consideration.
   a) If the number of harmonics mitigated for the best population is equal to the number of harmonics to be eliminated, end of execution flag is set.
3. A control sentence keeps executing $(i+1)^{th}$ iteration or skip it, leading the code to termination, according to end of execution flag.

This new control structure is represented in Fig. 5.9.

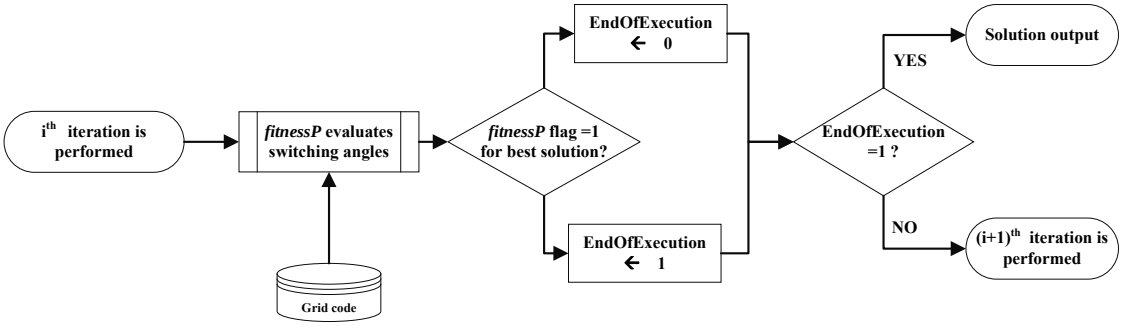


Figure 5.9 Decision logic to be implemented.

In order to provide a graphical representation of final code execution sequence after the changes above been implemented, a flow diagram is provided to the reader in Fig 5.10.

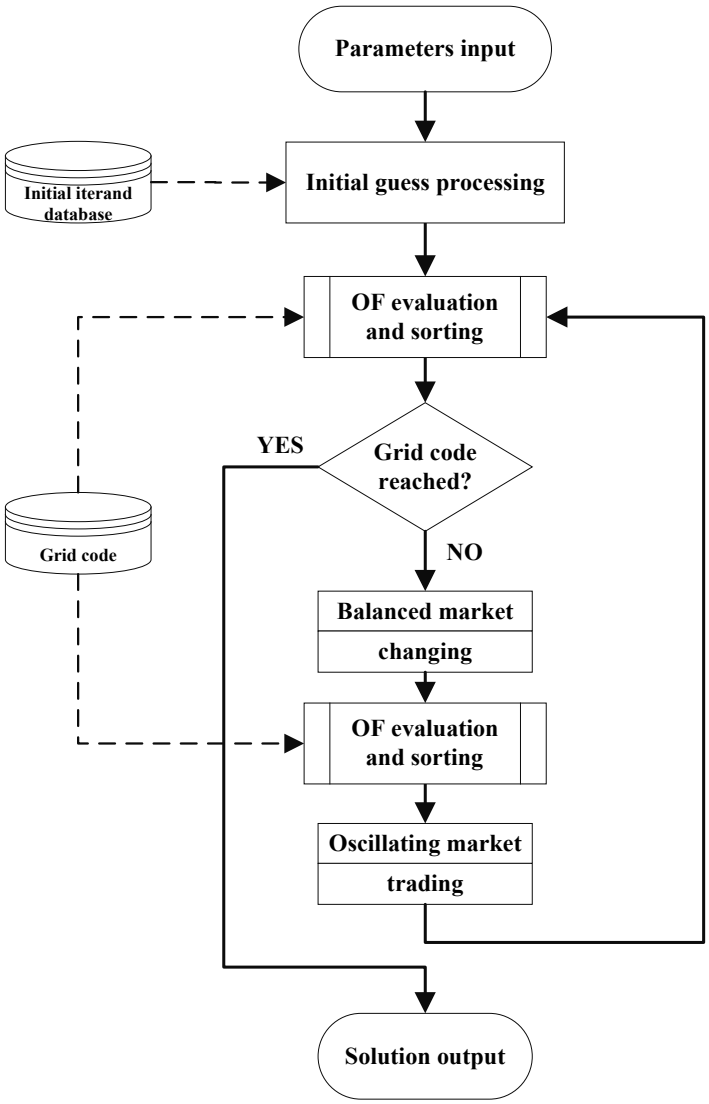


Figure 5.10 Code flow diagram.

After implementing these changes, a new batch of tests is proposed. These tests aims to validate the changes already mentioned and to obtain a definite characterization on execution time and convergence. Results will be presented in the next section.

5.6 Results obtained for improved implementation of the algorithm

The next task is to provide a figure on code optimization after the implementation of the changes proposed above. Due to availability of initial iterands, for the rest of calculations 15 switching angles and 17 harmonics would be used as specification for the problem, while using the grid code presented in Table 1.1.

In addition, all results presented from now have been obtained by use of the workstation machine. This aims to obtain the lowest computation time possible, in order to exploit the possibilities of the present implementation of EMA. This switch will provide useful information into consideration for future implementations of the algorithm pursuing *online* application.

5.6.1 Time and OF characterization

First consideration is execution time. Tests showed the algorithm to quickly converge to a valid solution providing that initial guess have been set accordingly with the modulation index and objective function parameters. By implementation of a coarse mesh of solutions, algorithm convergence can be guaranteed while maintaining the execution steady.

Figure 5.11 represents the execution time for a cumulative of 250 executions of the algorithm, while keeping the modulation index steady at $m_a = 1.1$.

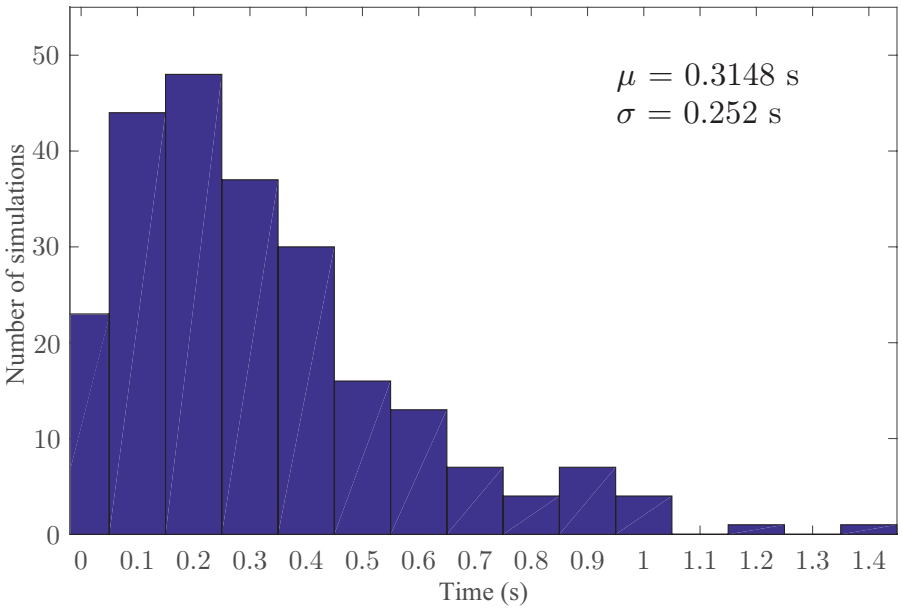


Figure 5.11 Distribution of execution time for 250 iterations, $m_a = 1.1$.

These results shows a trade-off between execution time, which is significantly lower, while keeping all the harmonics within the legislation. Statical analysis have arisen a mean time of execution of $\mu = 0.3148\,s$ and a standard deviation of $\sigma = 0.252\,s$, which lead to a 80% of executions to be completed before 0.5 seconds. This would mean having a compliant solution with the grid code in a mean time of 25 periods of a 50 Hz sinusoidal signal.

Cumulative representation can be also performed for the value of the objective function after the algorithm calculation. The lower the objective function value is, the lower the harmonic content presented on the output waveform. The representation for 250 execution with $m_a = 1.1$ is presented in Fig. 5.11).

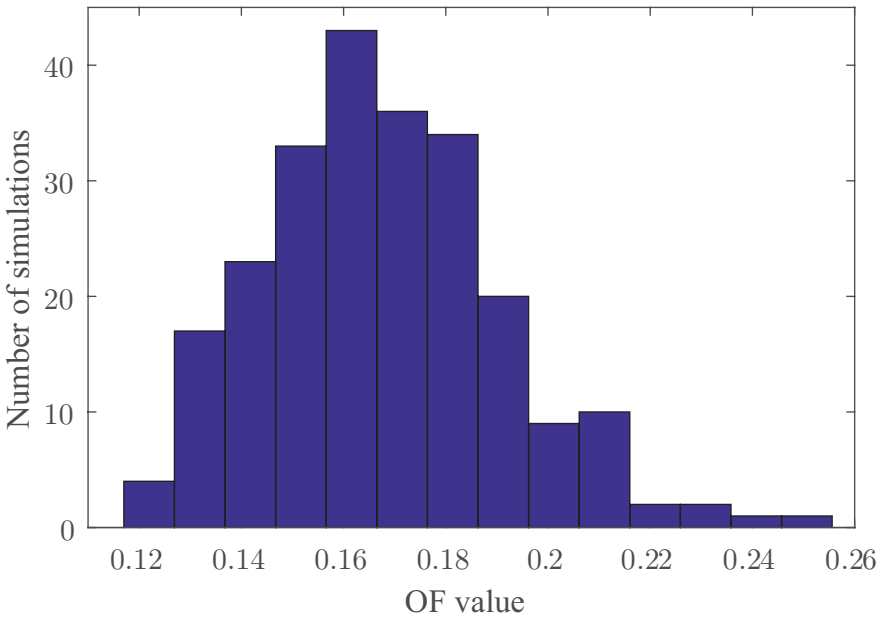


Figure 5.12 Value of objective function for SHM 15 switching angles / 17 harmonics with advanced features on workstation.

The mean value of the objective function for the data set is $\mu = 0.1678$, therefore showing the algorithm optimization is being performed successfully showing the convergence to the compliant solution on all the executions.

The EMA algorithm to solve the SHM problem has been tested for a wide range of modulation index m_a from 0.75 to 1.2. As a result, the switching angles values for all m_a range have been represented in Fig. 5.13.

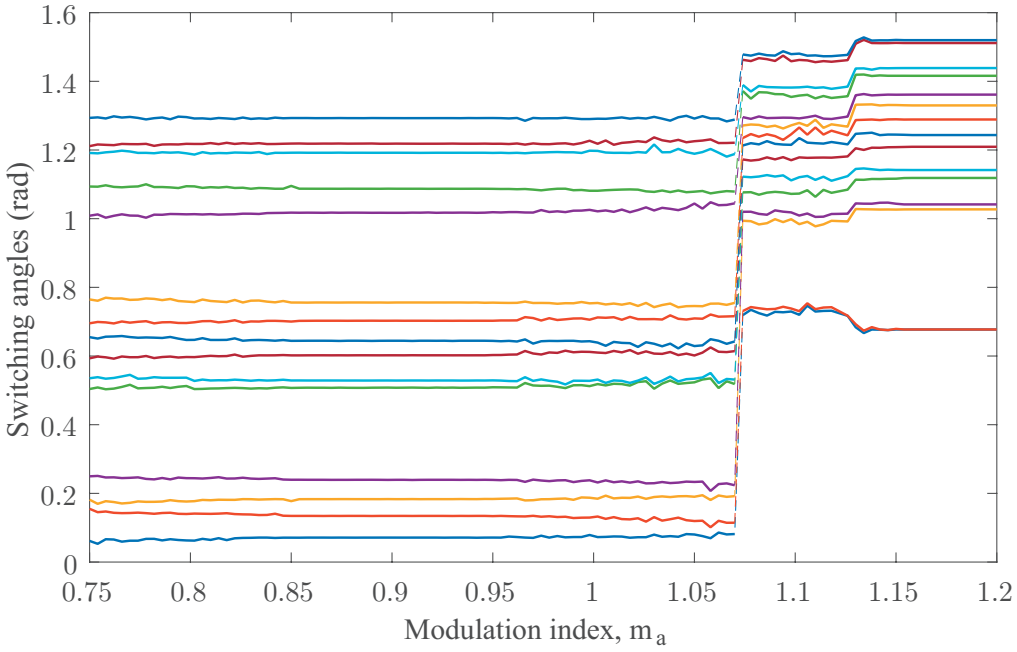


Figure 5.13 Switching angle values for a wide range of m_a between 0.75 and 1.2.

It can be observed that the switching angles sets are continuous except a step discontinuity between m_a 1.08 and 1.081, which can be associated for the algorithm to move from local minimums of the cost function as m_a is increased. This behaviour is very relevant in metaheuristic algorithm and contrast with the behaviour of gradient based methods. For the metaheuristic algorithm, it performs a proactive search, being able to consider solutions outside the actual local minima if it is expected to produce a relevant reduction in the cost function. Gradient based methods, however, would only displace in the direction of the gradient, therefore sinking the solution into the present minima.

5.6.2 Algorithm convergence

In order to address the issue of convergence, best solution found by the algorithm while reaching the final one has been analyzed. By sampling the best solution on several intervals, information as convergence rate, number of iterations required, and fulfillment of the solution for a transient period can be obtained. This information could be relevant for analyzing operation point changes on an online application.

This have been represented in Fig. 5.14, which shows the result for a calculation run at a $m_a = 1.0$ after convergence. Those harmonics which have not been below the limit are represented in one axis, while the second one represent the elapsed time of calculation, refereed to period of 50 Hz. Z-axis represents by a bar if the corresponding harmonic at the end of the period is compliant with the grid code. As it can be observed, upper order harmonics, which are subject of more strict grid code, are shown to reach convergence faster than lower order ones and fundamental.

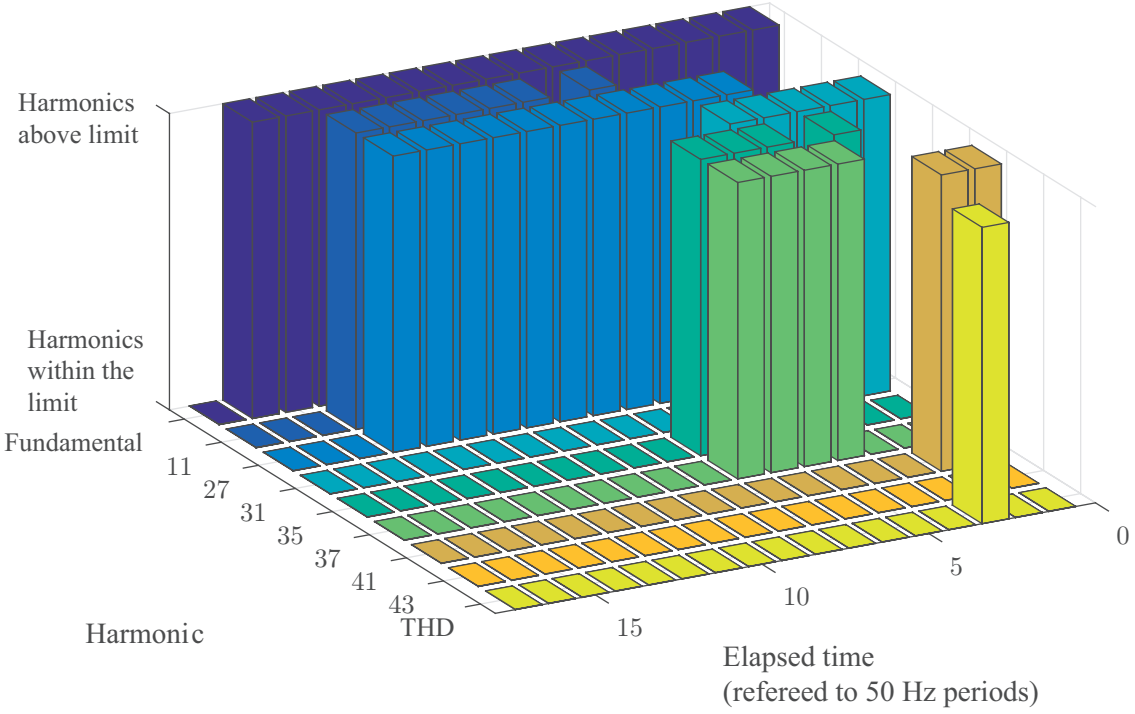


Figure 5.14 Convergence of calculations of EMA applied to SHM with $m_a = 1.0$, represented in windows of 50 Hz.

In order to expand the information provided by Fig. 5.14, the following representation has been proposed. Fig. 5.15, the harmonic content for each harmonic order is displayed along execution iterations with $m_a = 0.85$. In order to ease interpretation of the figure, harmonic contents are only displayed when the corresponding value is above the corresponding grid code limit. As it can be observed, most of the harmonics have been successfully mitigated below 100 miliseconds, while requiring a larger number of iterations for some higher order harmonics to converge.

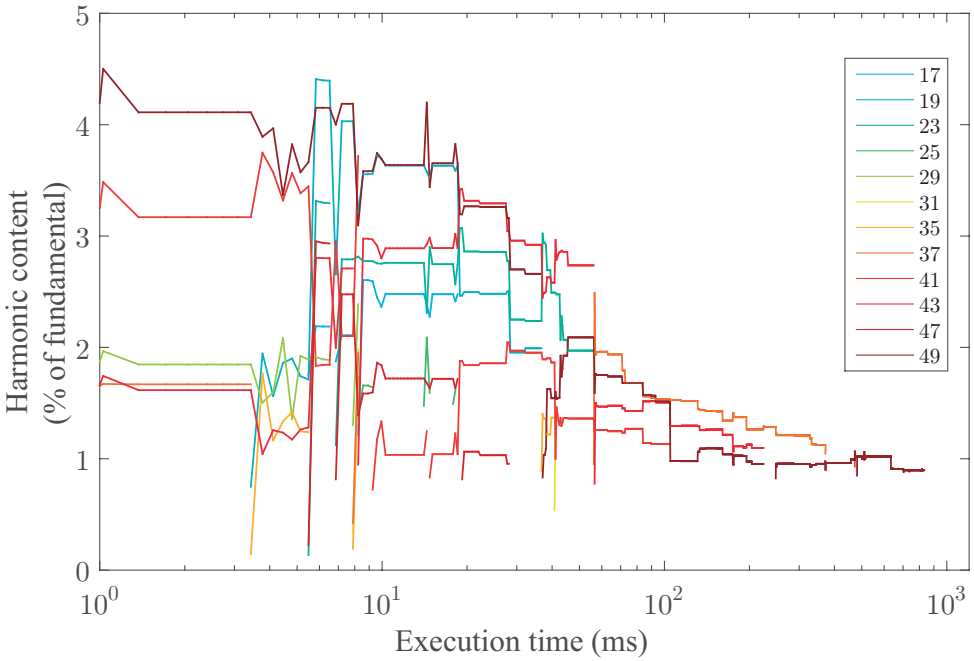


Figure 5.15 Time convergence of calculations of EMA applied to SHM with $m_a = 0.85$.

5.6.3 Simulation results

In order to validate switching angle calculation, several tests have been carried out with the simulation of a three-phase H-bridge inverter under Simulink-SimPowerSystem environment. Computed solutions have been fed into and time-domain simulation results have been examined. All the obtained switching angle values for a wide range of modulation index have been tested in order to confirm the fulfillment of the grid codes. As an example, in Fig. 5.16 it is represented the harmonic spectrum applying the obtained switching angles with a modulation index equal to $m_a = 0.9$.

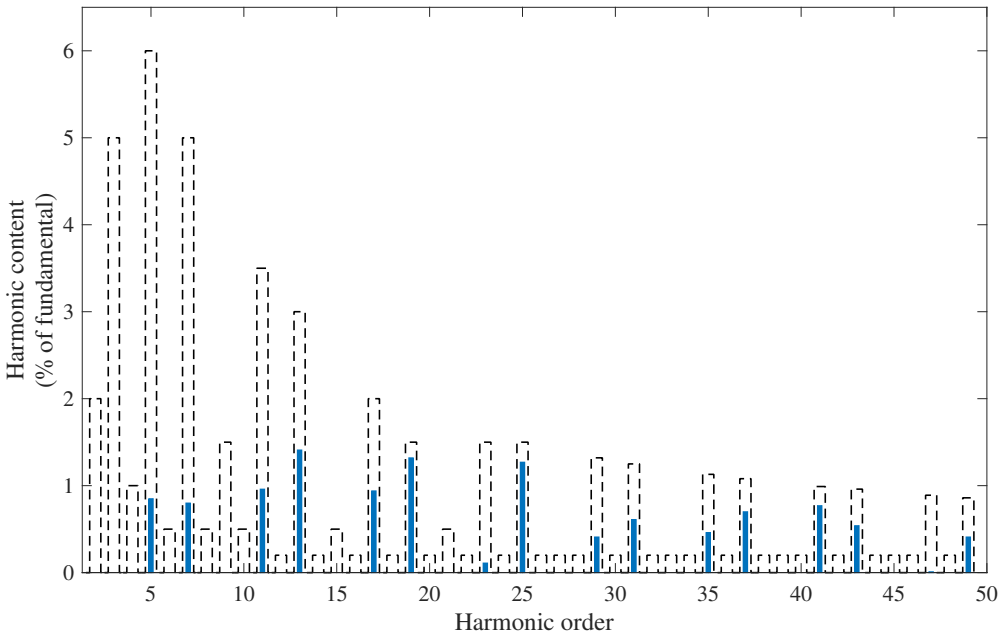


Figure 5.16 Harmonic content respect to fundamental for validating the obtained results, example with $m_a = 0.9$.

It can be observed that the computed waveform pattern complies with the grid code. In order to test all the obtained results, Fig. 5.17 represents the maximum harmonic content for each component on a sweep of modulation index between 0.75 to 1.2. It can be observed that the worst case for each component complies with the grid code.

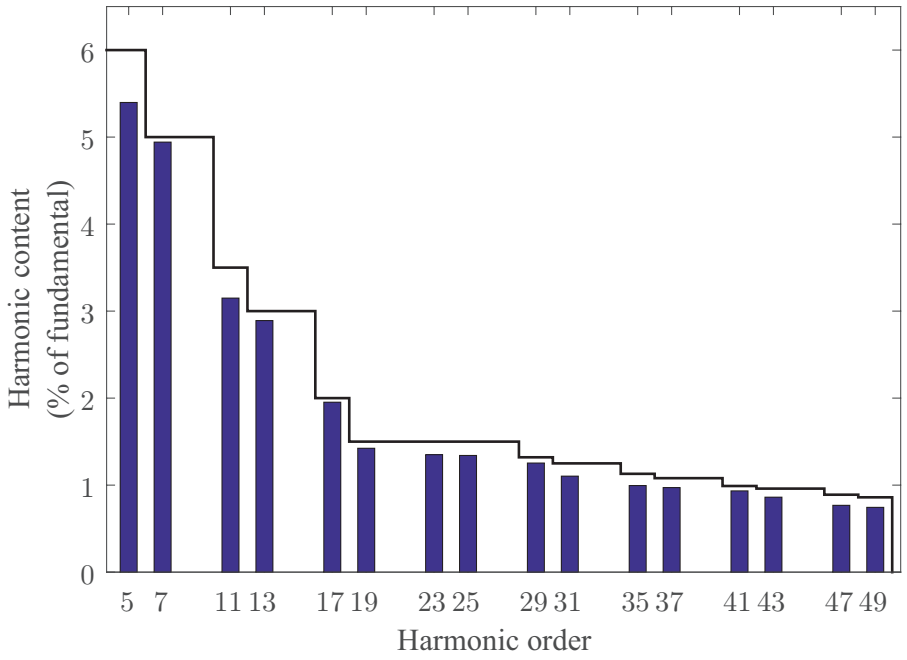


Figure 5.17 Maximum harmonic content respect to fundamental for validating the obtained results, m_a between 0.75 and 1.2.

5.7 Development of an user interface

As a request for the proposed work, the development of an user interface has been implemented by use of MATLAB GUI tools. User interface aims to reduce workload for the non-expert user, which is only required to modify execution parameters as m_a, grid code or a specific initial iterand, while keeping the launching and configuration of algorithm functions hidden. This way, the learning process required for executing the algorithm is drastically reduced and code sustainability is increased. The interface design can be seen in Fig. 5.18.

The main entry point for launching the program is found at `main.m` file. When launched from the `main.m` file, user is prompted to enter the problem definition, in terms of modulation index, numbers of harmonics into mitigation and number of switching angles to be calculated. Several options are, therefore, set by default, as for example the grid code and the initial iterand. The user can, however manually modify those values by selecting a box with these options. Grid code and initial iterand are parsed as a string of characters, which can be separated with spaces, tabs or commas. A button will display allowing the user to confirm the selection. The menus for modifying the initial iterand and grid code are shown in Fig. 5.19**(a)** and 5.19**(b)**, respectively.

The execution of the algorithm can be called by clicking the **Begin calculation** button, which will:

1. Parse inputs and display warning because of lack of consistency -values of number of switching angles or harmonics or m_a outside the expected region, not numerical input, etc.
2. Pass the information to EMA function.

3. Await for information to be calculated, displaying to the user the status of calculation.
4. Receive the calculated switching angles and execution data from the EMA function
5. Display the solution on the box and runtime information -time of calculation and value of OF.

The following information will be available for the user at any time:

- **Execution info**, as for example program state (*Idle* or *Calculating*), and elapsed time of the last execution of the program, displayed in seconds.
- **Output switching angles**, displayed in a box environment so that the user can export the data by using 'Copy' keyboard shortcuts (*Ctrl+C* on Windows environment).
- **Value of the OF** for the last execution of the program.
- **A plot of harmonic content** for the calculated solution, which can be expanded by double clicking on the figure and collapsed by clicking on 'Close plot' button. An example is represented in Fig. 5.20

The interface after execution displaying to the user the calculated results can be seen in Fig. 5.21

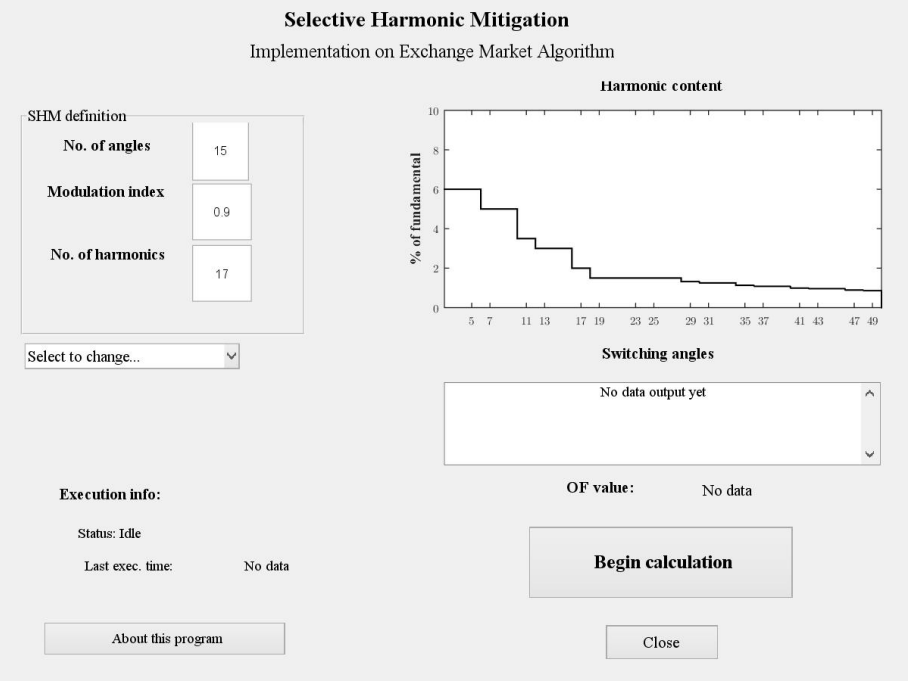
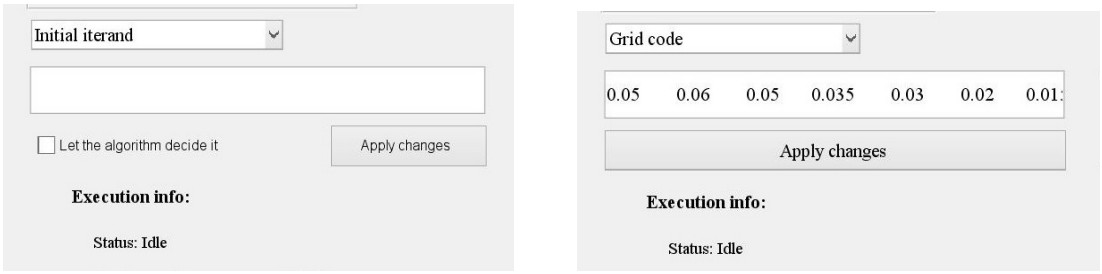


Figure 5.18 Interface developed for handling EMA algorithm.



(a) Initial iterand seleccion menu. **(b)** Grid code selection menu.

Figure 5.19 Detail view of configuration menus.

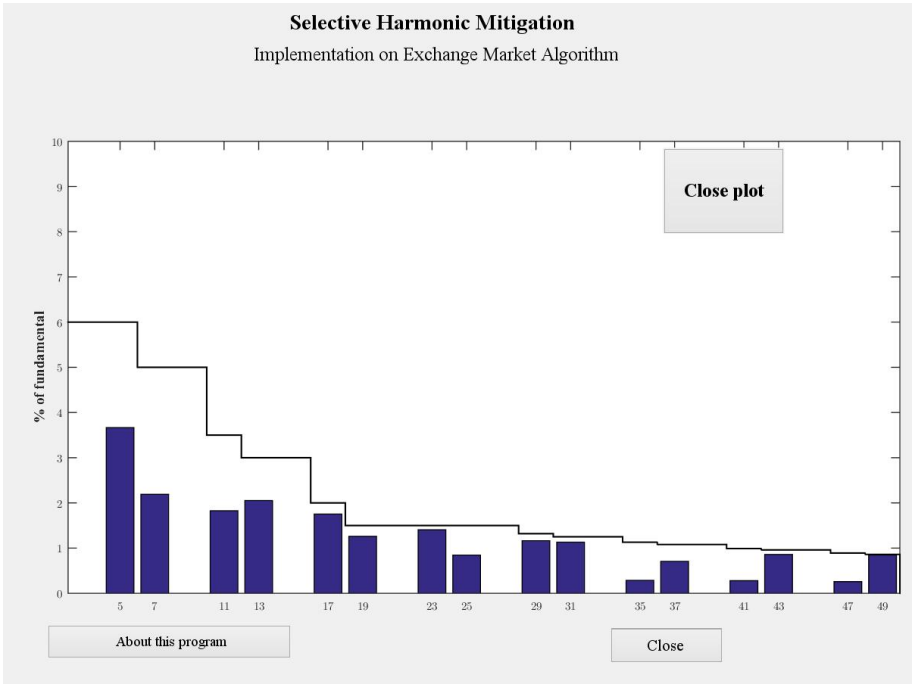


Figure 5.20 Representation of harmonic content.

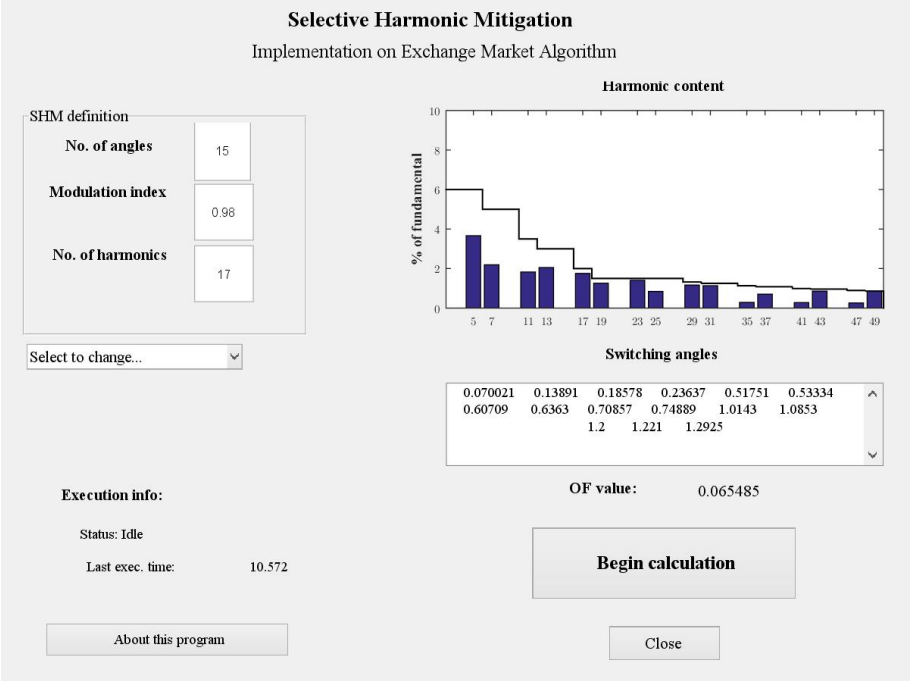


Figure 5.21 Interface for EMA after calculation.

6 Exchange Market Algorithm y su aplicación al SHM

El presente capítulo tiene como objetivo presentar un nuevo algoritmo heurístico para la solución del problema SHM, el Exchange Market Algorithm. Este algoritmo fue presentado en 2012 por Naser Ghorbani y Ebrahim Babaei en [29], habiendo demostrado un buen rendimiento en términos de número de iteraciones y convergencia, en comparación con otros métodos (GA, PSO, etc.) [44].

El presente capítulo pretende introducir brevemente los detalles técnicos de funcionamiento del algoritmo, así como mostrar los pasos seguidos en su implementación en MATLAB para el problema de SHM. De igual forma, se presentarán resultados de pruebas de rendimiento, en materia de tiempo, convergencia, contenido harmónico conseguido, etc. Por último, se introducirá la interfaz de usuario que ha sido diseñada con motivo de facilitar la manipulación por parte del usuario.

6.1 Descripción del algoritmo

El algoritmo EMA toma su inspiración del comportamiento de los accionistas operando en un mercado bursátil. En un mercado real, el comportamiento de los inversores se rige en grandes rasgo bajo dos ideas [43]:

- Los accionistas buscan incrementar el beneficio reportado por su cartera de acciones
- La demanda de acciones es factor fundamental en su precio, por lo cual la evolución del mercado dependerá de la compra y venta de estas acciones.
- Según las acciones de cada accionista, esté asumirá un mayor riesgo -accionistas peor posicionados- o será más conservador en sus acciones -accionistas mejor posicionados.
- La estabilidad del mercado afecta a la forma en la que se realizan las transacciones bursátiles. Mercados estables darán valores de acciones poco fluctuantes, pero limita las ganancias.

Esta filosofía fue interpretada por los autores como una estrategia poderosa para la resolución de problemas de optimización no lineal. En el paralelismo establecido entre el

problema bursátil y el problema de optimización, cada posible solución del problema se trata como un accionista. Los beneficios que reportan estas acciones se trasladan al valor que toma función objetivo para la solución en cuestión.

Para la evolución de las potenciales soluciones, se distinguen dos modos [29]: Por un lado, un **modo no oscilante**, el cual imita a un mercado estable, donde se lleva a cabo procesos de cambio de acciones entre accionistas. Por otro lado, el **modo oscilante** se asocia con mercados variables, en los cual se realizan compra-venta de acciones, con una importante componente de riesgo e incertidumbre.

Con motivo de plasmar las diferencias de comportamiento en función del posicionamiento de cada accionista en el mercado, se establecen tres disticiones en las soluciones: **soluciones de alto rango**, las cuales no sufren cambios; **soluciones de mediano rango**, las cuales buscan elevar su posición en pequeña escala, y **soluciones de bajo rango**, las cuales se ven afectada por procesos evolutivo más severos que los de mediano rango.

La implementación en particular de cada uno de los mercados y las ecuaciones que rigen los procesos de cambios, así como los posibles ajustes que permite el algoritmo, están recogidas en [29]. Con la adición de unas estructuras de control adecuadas, el algoritmo puede implementarse en un lenguaje de programación.

Un pseudocódigo de la implementación del programa se presenta a continuación, así como en Fig. 5.1.

1. Se genera una población inicial de soluciones.

2. Mientras no se haya alcanzado la condición de parada,

1. Se calcula el valor de la función objetivo para todas las posibles soluciones en la población.
2. Los valores obtenidos en 1., se emplean para ordenar dichas soluciones y clasificarlas según su posición.
3. Se realizan los procesos de mercado balanceado:
   1) Los *accionistas de mediano rango* (soluciones) realizan un proceso de *cambio* de acciones en modo no oscilante.
   2) Los *accionistas de bajo rango* (soluciones) realizan un proceso de *cambio* de acciones en modo no oscilante.
4. Se calculan los valores de la función objetivo y se ordenan la población en función de los valores, como en 1. y 2.
5. Se realizan los procesos de mercado oscilante:
   1) Los *accionistas de mediano rango* (soluciones) realizan un proceso de *compra-venta* de acciones en modo oscilante.
   2) Los *accionistas de bajo rango* (soluciones) realizan un proceso de *compra-venta* de acciones en modo oscilante.
6. Salto a **2.**.

3. Se ha encontrado la mejor solución dentro de los criterios establecidos

Este algoritmo ha demostrado una mejora en cuestión de tiempo frente a otros algoritmos en el estado del arte, así como su aplicabilidad a problemas de ámbitos diversos, como se puede ver en [44] y [42]. Una comparación de estos resultados se presenta en la Tabla 5.1.

6.2 Implementación en MATLAB

En esta sección se presenta una implementación de EMA para su aplicación al SHM realizada en MATLAB. Se ha elegido este entorno por el acto carácter vectorial con el que está definido. Futuras aplicaciones pueden realizarse en herramientas de bajo nivel con motivo de aprovechar las capacidades de hardware más profundamente.

Para la implementación de SHM, se ha montado la función de costes propia del problema sobre el código optimizado de los autores. Esta función objetivo permite considerar un número determinado de armónicos en mitigación, incluyendo el fundamental, así como la distorsión armónica total (THD) para estos armónicos. Otras consideraciones pueden realizarse en la función objetivo, por ejemplo aspectos económicos, debido a la dependencia existente entre el precio de los filtros de salida y el contenido armónico en las formas de onda generadas.

El código se estructura en una serie de funciones, con motivo de separar las diferentes actuaciones. Así, `EMA.m` proporciona la función principal para la ejecución, la entrada de parámetros del usuario y la presentación. La función objetivo del problema se implementa en `fitnessN.m`. La evolución de la población se realiza en las funciones `oscillation.m` y `notoscillation.m`, correspondientes a ambos métodos. Finalmente, `Initial.m` y `SHM.m` proporciona parámetros necesarios para el resto de funciones. Una representación esquemática puede obtenerse en Fig. 5.2

La configuración por defecto que se establece para los cálculos realizados ha sido: 15 ángulos de cortes, mitigación de harmónicos hasta el 49º, y límites de distorsión armónica definidos en EN 50160 + CIGRE WG 36-05 (Tabla 2.1). Los resultados presentados están realizados especialmente en el rango de m_a entre 0.75 y 1.2, debido a la disponibilidad de iterandos iniciales para este rango.

6.3 Resultados obtenidos en la implementación inicial en MATLAB

En esta sección se recogen una serie de pruebas realizadas con una implementación inicial del algoritmo, sin el uso de mecanismos especiales de control ni iterandos iniciales específicos. Estas pruebas han sido realizadas en el equipo portátil, con motivo de facilitar la depuración de código. La configuración del problema para estas pruebas concretas ha sido de: 6 ángulos de corte para mitigación de 7 armónicos, incluyendo el fundamental y THD.

Los resultados obtenidos se presentan mediante gráficas de valores acumulados, con motivo de obtener una distribución significativa de los valores. Los resultados se han obtenido para la distribución de tanto el tiempo de ejecución como del valor de la función objetivo, para un $m_a = 1.1$ y un número de iteraciones de 500 (Fig. 5.3), de 1000 (Fig. 5.4) y de 3000 (Fig. 5.5).

Estos resultados permite observar que a mayor número de iteraciones, el tiempo de ejecución crece prácticamente linealmente y el valor de la función objetivo tras la ejecución decrece y se distribuye más centrado en la media. Un resumen de los resultados obtenidos se presenta en la siguiente tabla:

Table 6.1 Resultados estadísticos de las pruebas iniciales.

Nº de iteraciones	Tiempo	Valor de OF
500	Percentil 90: 0.874	Percentil 80: 0.523
1000	Percentil 90: 1.836 s	Percentil 90: 0.759
3000	Percentil 90: 5.29 s	Percentil 90: 0.438

De los datos obtenidos se puede extraer que, entre los valores de 1000 y 3000 se consigue mitigar suficientemente el valor de los armónicos en consideración, sin embargo es necesario establecer secuencias de control avanzadas. Estas secuencias deberán considerar además del número de iteraciones, el cumplimiento del *grid code*, con motivo de conseguir un equilibrio entre el contenido armónico y el tiempo de ejecución.

Una última consideración es estudiar cómo evoluciona el valor de la función objetivo con cada una de las iteraciones. Dentro de la formulación del EMA, cada iteración presentará un valor igual o menor de la función objetivo, que en el caso de SHM (véase ec. 2.5) se desea reducir a valores próximos a cero cuando se haya alcanzado el cumplimiento con el *grid code*.

Esta representación se presenta en Fig. 5.6. Como se puede observar, EMA es capaz de reducir considerablemente el valor de la función objetivo en los primeros cientos de iteraciones. Sin embargo, requiere de un tiempo mayor para reducir a partir de éstas, debido a la convergencia de algunos armónicos de orden superior.

Un breve resumen de lo obtenido en este apartado se presenta a continuación:

- EMA permite una convergencia rápida en términos de iteraciones.
- El número de iteraciones presenta sensibilidad respecto del iterando inicial, por lo que es necesario el empleo de iterandos iniciales más precisos.
- EMA consigue el cumplimiento del *grid code* en un número de iteraciones en general menor a las establecidas, por lo que se propone emplear ese cumplimiento como control del número de iteraciones.

6.4 Análisis de rendimiento del código

Con motivo de estudiar la ejecución de código en profundidad, se han empleado las herramientas de análisis de rendimiento (en inglés, *profiling*) incluidas en MATLAB. Para estas pruebas, se ha partido del código configurado para los casos anteriores, y se ha estudiado los resultados en materia de tiempo de ejecución obtenidos para el código completo y para funciones particulares del mismo. Se hace notar que, aunque los resultados obtenidos difieren de la realidad al no disponer de mecanismo de optimización en *tiempo de ejecución* (conocidos por *JIT*), éstos permiten el estudio comparativo entre elementos del código.

El estudio comparativo de funciones de alto nivel del programa se presenta en Tabla 5.2 y en Fig. 5.7. En la tabla se incluyen: número de llamadas a la función, tiempo total de

ejecución o *total time* y tiempo propio de la función o *self time*. Una explicación somera de ambas funciones es la siguiente: el tiempo total engloba el tiempo entre que la función ha sido llamada hasta que finaliza su ejecución; Por contra, el tiempo propio considera exclusivamente el tiempo que la función se encuentra en primer plano, excluyendo así el tiempo dedicado a las subfunciones contenida en ésta. Más información puede obtenerse en [45].

Como se puede observar, la función `fitnessN.m` es la responsable de más del 75% del tiempo de ejecución, debido al elevado número de operaciones no lineales que la forman. Es también relevante la aportación de las funciones `oscillation.m` y `notoscillation.m`, las cuales se componen de operaciones algebraicas y generación de números aleatorios, mientras que otras funciones aportan tiempo despreciable respecto al total.

Empleando el profiler sobre la función `fitnessN.m`, se han obtenido los resultados presentados en Tabla 5.3 y Fig. 5.8. Se observa que la evaluación de cada armónico y la suma de dicho contenido a la función de costes constituye la mayor parte del tiempo de ejecución. Los resultados obtenidos permiten apreciar las capacidades de MATLAB de proveer mecanismos para la vectorialización de código, especialmente para un número de datos elevados.

6.5 Implementación de funciones avanzadas en EMA

En este capítulo se pretende presentar la implementación de las dos cuestiones que fueron propuestas en las pruebas y análisis realizados con anterioridad.

Como solución a la rígida dependencia del iterando inicial del problema SHM, se realizado la implementación de una base de algoritmos iniciales, en función de m_a. Esta base datos, implementada en `initial_pop` permite que la solución inicial se encuentre en un rango cercano de la final, reduciendo drásticamente el tiempo de ejecución de algoritmo. Estas soluciones iniciales pueden obtenerse mediante exploración de forma gradual para un rango de m_a concreto. Estos datos se han obtenido de trabajos realizados en algoritmos SHM, como los realizados por Patricio López bajo la dirección de Ph.D. Ramón Portillo Guisado y en [20].

Para proveer de un método eficaz para finalizar la ejecución del algoritmo cuando una solución satisface el *grid code*, se ha propuesto el siguiente modificación a diagrama de flujo, representada en Fig. 5.9:

1. Se realiza la iteracion i-ésima.
2. Una versión modificada de `fitnessN`, `fitnessP`, evalúa la función de coste y proporciona una variable que indica si dichos ángulos de corte cumplen las condiciones establecidas en el *grid code*
   - **a)** Si la mejor solución cumple con el *grid code*, se marca el fin de la ejecución
3. La secuencia de control finaliza la ejecución del algoritmo si se marcó el fin de la ejecución, o en su defecto continúa con la iteración i+1-ésima.

El nuevo diagrama de flujo del programa se presenta en Fig. 5.10.

6.6 Resultados obtenidos con la implementación mejorada del algoritmo

Es esta sección se recogerán los resultados obtenidos con la nueva implementación del algoritmo. Debido a la disponibilidad de iterandos, las pruebas se han realizado para el problema de 15 ángulos de corte y 17 armónicos. De igual forma, las pruebas se han realizado en la máquina *workstation*, con motivo de estudiar los límites en los que el tiempo de ejecución puede ser reducido. Estas consideraciones proporcionarán información útil sobre futuras implementaciones en sistemas que realicen el control *online*.

6.6.1 Caracterización del tiempo y el valor de OF

En primer lugar se considerará el tiempo de ejecución del algoritmo. Las pruebas realizadas muestran que el algoritmo converge a una solución válida en tiempo reducido habiéndose provisto de un iterando inicial ajustado al valor de m_a.

Los resultados acumulados de 250 ejecuciones, para un valor de $m_a = 1.1$ se muestran en la Fig. 5.11, de donde se extra un valor medio $\mu = 0.3148\,s$ y desviación típica $\sigma = 0.252s$, o lo que es lo mismo, se consigue de media obtener una solución en 25 periodos de 50 Hz. De igual forma, el valor de la función objetivo para este caso se ha recogido en la Fig. 5.12, la cual presenta un valor medio de $\mu = 0.1678$.

De los resultados presentados, se puede observar el compromiso conseguido entre tiempo de ejecución, significativamente reducido, con el cumplimiento de la legislación en materia de armónicos. De igual forma, el valor de la función objetivo muestra que el algoritmo es capaz de conseguir la convergencia en todas las ejecuciones.

Ejecutándo el algoritmo para un rango de m_a entre 0.75 y 1.2, se han generado los ángulos de corte representados en Fig. 5.13. Es relevante el hecho de que, en el rango de m_a entre 1.08 y 1.081, se observa una discontinuidad en el valor de los ángulos de corte. Este fenómeno se asocia a la evolución del algoritmo entre mínimos locales, con motivo de mejorar el valor de la función objetivo. Este hecho asegura la capacidad de evitar el estancamiento en un mínimo local, como ocurre en el caso de algoritmos que emplean el gradiente en su evolución.

6.6.2 Convergencia del algoritmo

Para evaluar la convergencia del algoritmo, se ha analizado las soluciones intermedias obtenidas por el algoritmo en el proceso de cálculo. Esta información proporciona información de utilidad para caracterizar la evolución del algoritmo ante transitorios en una implementación *online*.

Empleando esta idea, se ha represando en Fig. 5.14 los resultados obtenidos para la ejecución con $m_a = 1.0$. Para facilitar la representación, sólo se han mostrado los armónicos con valor no nulo en las ventanas de 50 Hz en las que se han observado los valores. El eje Z representa si dicho armónico cumple o no con las especificaciones del *grid code*. Es posible observar que, mientras que los armónicos de orden superior convergen más rápido que aquellos de orden inferior y que la componente fundamental.

Empleando este mismo concepto, se ha representado en Fig. 5.15 la evolución para cada iteración del contenido armónico presente en cada armonico, en el caso de $m_a = 0.85$, habiéndose representado únicamente los valores que exceden el valor máximo. Como se

puede observar, la mayoría de los armónicos alcanza son mitigados antes de 100 milisegundos, mientras que un número reducido requiere un mayor número de iteraciones para converger.

6.6.3 Resultados de simulación

Con motivo de validar los resultados calculados, se han realizado la simulación de un inversor en puente H trifásico en Simulink-SimPowerSystems a partir de los resultados obtenidos de EMA. Los valores de los ángulos de corte han demostrado cumplir los límites establecidos en el *grid code*. Como ejemplo, los resultados obtenidos para el cálculo para $m_a = 0.9$ se han presentado en Fig. 5.16.

De igual forma, se ha representado en Fig. 5.17 el máximo valor obtenido para cada uno de los armónicos en el rango de m_a entre 0.75 y 1.2.

6.7 Desarrollo de una interfaz de usuario

Dentro de uno de los requisitos del presente trabajo, se ha desarrollado una interfaz de usuario empleando las herramientas de MATLAB. Ésta pretende reducir la carga de trabajo de usuarios no familiarizados con el algoritmo, permitiendo configurar el mismo de forma sencilla, a la vez que evitando al usuario manipular directamente a nivel de código.

La interfaz de usuario se ejecuta a partir de la función `main.m`. Desde ésta (Fig. 5.18, el usuario podrá introducir los parámetros del problema -m_a, número de ángulos de corte y de armónicos- y modificar los valores predefinidos de iterando inicial o de *grid code* (Fig. 5.19).

Con los resultados introducidos, la interfaz realizará la comprobación de los valores introducidos, ejecutará el EMA con estos parámetros, procesará los resultados, y proporcionará la información al usuario. Esta información engloba: tiempo de ejecución, estado de la ejecución -*parado* o *en ejecución*-, ángulos de corte resultantes, valor de la función objetivo y una representación gráfica del contenido de los armónicos comparados con los límites para cada uno.

7 Conclusiones y Trabajos Futuros

En relación al algoritmo CUSIMANN y su implementación para el problema de Mitigación Selectiva de Armónicos, gracias a los resultados de *profiling* y de modificación de parámetros de compilación se ha conseguido obtener información de gran importancia para futuros trabajos en el campo de los algoritmos metaheurísticos. Algunos de estos puntos de vital importancia son: el aprovechamiento del ancho de banda de memorias, los límites en los recursos de hardware existentes, etc. De igual forma, se ha estudiado la posibilidades de mejora para futuras implementaciones. Mediante la gestión de las opciones de optimización del compilador así como la elección de la versión del mismo coherente con la placa en cuestión permite obtener cambios importantes en los tiempos de ejecución

En relación a la investigación desarrollada en el Exchange Market Algorithm y su implementación para el problema de Mitigación Selectiva de Armónicos, se han obtenido resultados prometedores. Las pruebas documentadas han demostrado que, en materia de convergencia y tiempo de ejecución, EMA se postula como una interesante alternativa para el problema SHM. De igual forma, mediante las simulaciones llevadas a cabo, se han podido validar las soluciones obtenidas, así como estudiar el comportamiento ante transitorios del mismo.

De igual forma, EMA ha lanzado la idea de una implementación *online* del SHM. Aunque ante aspectos no deterministas del algoritmo debido a la aleatoriedad existente en el cálculo de soluciones, se abre interesantes lineas de trabajo para su mejora. Por ejemplo, mediante paralelización o redefiniendo la selección de iterandos iniciales, pueden llegar a reducirse esta variabilidad existente.

La labor de investigación realizada ha dado lugar a la presentación del *paper* Selective Harmonic Mitigation Technique Based on the Exchange Market Algorithm for High-Power Applications, con autores Francisco J. González, Marta Laguna, Abraham Marquez, Jose I. Leon, Ramon Portillo y Leopoldo G. Franquelo, presentado con motivo de la 43ª Conferencia Anual de la Sociedad de Electrónica Industrial (IECON 2017), celebrado en Beijing, China. Este *paper* se encuentra en revisión para su aceptación.

7.1 Líneas futuras de investigación

En relación al Exchange Market Algorithm y su implementación con el problema de Mitigación Selectiva de Armónicos, las líneas de investigación a seguir en un futuro

buscarán explotar las nuevas oportunidades que se abren con la reducción en el tiempo conseguida.

Una línea de trabajo a seguir es la implementación de un inversor montando el método SHM con cálculo de los ángulos de corte en tiempo real. En el estado del arte actual, no se ha presentado ninguna implementación de este tipo de cálculo, reduciendo SHM a una implementación *offline* donde los ángulos de corte se calculan previamente. Esto incrementa los costes de desarrollo del convertidor y una reducción de las capacidades para adaptarse a cambios en los parámetros.

Sin embargo, un aspecto de vital importancia para las implementaciones *online* es el tiempo de cálculo. Dependiendo de la legislación aplicable, las medidas a realizar durante transitorios se realizan a partir de la media obtenida para varios periodos de señal. De esta forma, se permite que el contenido armónico exceda los límites durante un número reducido de periodos. Esto obliga al convertidor a alcanzar los límites establecidos en la legislación en un máximo de periodos, en el orden de 10 a 20 periodos [46].

Sin embargo, no solo es necesario considerar el tiempo de ejecución. Es importante considerar el tiempo que transcurre desde que se finaliza el cálculo, se escribe en el bus (dedicado o compartido, implementado en TCP-IP o I^2C, etc.), es recibido por el convertidor, se descodifica y se transmite al *gate driver*. Este tiempo dificulta la consecución de los límites para respuesta ante cambios en el convertidor.

Con el objetivo de elaborar un prototipo del convertidor, se ha propuesto realizar una implementación en el ecosistema dSPACE. dSPACE se constituye como una plataforma para el diseño rápido de prototipos en un amplio rango de campos (automovilístico, aeroespacial, industrial, etc.). De igual forma, proporciona herramientas para el control de motores y validación y verificación de sistemas bajo hardware-in-the-loop (HIL). dSPACE permite programar estas características mediante herramientas de alto nivel como Simulink [47].

Una implementación del EMA bajo el hardware de dSPACE, específicamente en el equipo DS1007 PPC Processor Board y EV1048 I/0 Expansion Board [48] se está estudiando en la actualidad. El equipo dSPACE y el convertidor en el que será montado se representa en la Fig. 7.1

(a) Equipo dSPACE. **(b)** Convertidor.

Figure 7.1 Equipo propuesto para las pruebas experimentales.

Es también necesario mencionar la posibilidad de implementar el algoritmo EMA en un paradigma de programación diferente. Trabajos futuros pueden considerar la factibilidad de

diferentes plataformas y lenguajes, en busca de mejorar la secuenciación de los recursos de cálculo o reducir las capacidades del hardware necesario para la ejecución del algoritmo.

Es necesario enfatizar el carácter de MATLAB como lenguaje interpretado. Aunque esto permite disponer de un mayor número de recursos para la vectorialización y el manejo de gran número de datos, el código en MATLAB se ejecuta bajo una máquina virtual, la cual interpreta el código fuente durante la ejecución del mismo. Aunque a día de hoy se han implementado nuevos mecanismos para la optimización del código, el rendimiento de MATLAB ha sido superado por el obtenido por lenguajes compilados. como por ejemplo C, C++ o Fortran [35] [49]. Posibles revisiones del código podrían llevar a la implementación del mismo en lenguajes de bajo nivel, permitiendo así medir la dependencia del rendimiento con el lenguaje de programación.

Otro tema de importancia a cubrir es la implementación del algoritmo en un paradigma de programación diferente. La paralelización de algoritmos metaheurísticos ha sido un campo de estudio muy relevante [24] [28], debido al importante número de tareas que pueden ejecutarse con independecia del resto. Trabajos futuros podrían considerar la implementación de EMA en sistemas de procesamiento paralelo, como por ejemplo CUDA™ o las soluciones multiprocesador de Intel™.

8 Conclusions and Futures Work

Related to CUSIMANN implementation for Selective Harmonic Mitigation, profiling results and compilation parameters provided useful information for future work in the field of parallel metaheuristic algorithms. Important topics as focusing on memory bandwidth, with larger transfers, and on code occupancy -availability of resources, etc.- has been documented. In addition, the optimization possibilities for future implementation have been measured. Proper optimization settings and coherence between compiler version and board have to be reached in order to get the best performance.

On the research performed on implementing Exchange Market Algorithm for the Selective Harmonic Mitigation, promising results have been obtained. Time and convergence levels reached by the algorithm have proven EMA as a valid alternative into consideration for SHM problem. In addition, simulation results have also provided information for validating its calculated solutions and characterization of its transient behaviour.

Exchange Market algorithm has also empowered the idea of a *online* implementation of SHM solution. While non-deterministic behavior of the algorithm due randomness dependence of solver, it opens interesting working lines for improvement. For example parallelization or redefinition of the initial iterand mesh, may aim to reduce the variations on execution time and solutions.

The research labour performed on EMA originated the presentation of the paper Selective Harmonic Mitigation Technique Based on the Exchange Market Algorithm for High-Power Applications, authored by Francisco J. González, Marta Laguna, Abraham Marquez, Jose I. Leon, Ramon Portillo and Leopoldo G. Franquelo, submited for 43th Annual Conference of the IEEE Industrial Electronics Society (IECON 2017) held in Beijing China. Paper is under revision for acceptance.

8.1 Future Working Lines

In relation to Exchange Market Algorithm and its implementation to the Selective Harmonic Mitigation problem, future working lines have to be established in order to exploit the achieved reduction in execution time.

One important research line to follow is the implementation of a power inverter mounting the SHM technique for *online* computation of the switching angles. Nowadays, there are no documented implementations of this methods, reducing SHM to an *offline* implementation, as switching angles are precomputed. Therefore, this causes an increase on the developing costs and a reduction on the flexibility to adapt to changes on the parameters.

However, main consideration for an *online* implementation is the time of calculation. Depending on the regulation, measurements on transient changes of output waveform are averaged on a defined number of periods, therefore allowing for harmonic content to exceed legislation for a reduced number of periods. Consequently, a power inverter is required to achieve harmonic compliance on a certain periods of grid signal. This value oscillates between 10 and 20 periods [46].

However, not only time execution is relevant in *online* application. It is also relevant to consider time since calculation has been finished, written onto a bus (dedicated bus or shared, under I^2C, TCP-IP, etc), received on the converter, decoded and transmitted to gate drives, as it would difficult achievement on established time constrains.

One interesting implementations have been proposed, for example under the prototype environment dSPACE. dSPACE provides a platform for designing prototypes for a wide ranges on applications (automotive, aerospace, industrial, etc.). In addition, it also provides support for motor drive interfaces and hardware-in-the-loop (HIL) verification and validation. All of these features can be programmed under high-level environments, as for example Simulink [47].

A proposal of implementing EMA code under the dSPACE hardware, specifically under DS1007 PPC Processor Board and EV1048 I/O Expansion Board [48] is on idea developing process nowadays. dSPACE control equipment and converter is shown in Fig. 8.1

(a) dSPACE unit. **(b)** Converter.

Figure 8.1 Proposed setup for experimental tests.

It is also worth considering the possibility of performing the implementation of EMA on a different programming paradigm. Future works may consider different platforms and languages for implementing the algorithm, in order to take the advantage of better sequencing of the calculation or reducing the required hardware resources for obtaining solutions.

It is necessary to point that MATLAB is an interpreted programming language. Whilst providing high-lever resources for vectorialization and handling large data sets, MATLAB code is executed under a virtual machine, which interprets source code at runtime. Even though new tools built into MATLAB environment allows code optimization, MATLAB performace has been overtaken by the obtained by use of compiled, as for example C, C++ or Fortan [35] [49]. New work could lead to implementations on low-lever languages, in order to benchmark the dependence on performance of the programming language.

Another relevant issue to cover is the implementation on a different programming paradigm. Metaheuristic algorithms have been a matter of study in the field of code parallelization [24] [28], due to the significative amount of tasks which can be executed independently from others. Further investigations could aim to implement EMA under parallel environments, as for example CUDA™ or Intel™ multi-core solutions.

Bibliography

[1] NVIDIA, *NVIDIA's Next Generation CUDA Compute Architecture. Fermi*.

[2] N. Mohan, T. M. Undeland, and W. P. Robbins, *Power Electronics. Converters, Applications and Design*. John Wiley and Sons, Inc, third ed., 2003.

[3] M. Yilmaz and P. T. Krein, "Review of battery charger topologies, charging power levels, and infrastructure for plug-in electric and hybrid vehicles," *IEEE Transactions on Power Electronics*, vol. 28, pp. 2151–2169, May 2013.

[4] S. Nandi, H. A. Toliyat, and X. Li, "Condition monitoring and fault diagnosis of electrical motors-a review," *IEEE Transactions on Energy Conversion*, vol. 20, pp. 719–729, Dec 2005.

[5] S. Kouro, M. Malinowski, K. Gopakumar, J. Pou, L. G. Franquelo, B. Wu, J. Rodriguez, M. A. Perez, and J. I. Leon, "Recent advances and industrial applications of multilevel converters," *IEEE Transactions on Industrial Electronics*, vol. 57, pp. 2553–2580, Aug 2010.

[6] J. M. Carrasco, L. G. Franquelo, J. T. Bialasiewicz, E. Galvan, R. C. PortilloGuisado, M. A. M. Prats, J. I. Leon, and N. Moreno-Alfonso, "Power-electronic systems for the grid integration of renewable energy sources: A survey," *IEEE Transactions on Industrial Electronics*, vol. 53, pp. 1002–1016, June 2006.

[7] G. de les Illes Balears. Conselleria de Comerç Indústria i Energia, *Pla Director Sectorial Energètic de les Illes Balears.*

[8] L. Gyugyi, "Unified power-flow control concept for flexible ac transmission systems," *IEE Proceedings C - Generation, Transmission and Distribution*, vol. 139, pp. 323–331, July 1992.

[9] J. Beerten, O. Gomis-Bellmunt, X. Guillaud, J. Rimez, A. van der Meer, and D. V. Hertem, "Modeling and control of hvdc grids: A key challenge for the future power system," in *2014 Power Systems Computation Conference*, pp. 1–21, Aug 2014.

[10] M. Liserre, T. Sauter, and J. Y. Hung, "Future energy systems: Integrating renewable energy sources into the smart power grid through industrial electronics," *IEEE Industrial Electronics Magazine*, vol. 4, pp. 18–37, March 2010.

[11] F. Blaabjerg, M. Liserre, and K. Ma, "Power electronics converters for wind turbine systems," *IEEE Transactions on Industry Applications*, vol. 48, pp. 708–719, March 2012.

[12] F. Blaabjerg, R. Teodorescu, M. Liserre, and A. V. Timbus, "Overview of control and grid synchronization for distributed power generation systems," *IEEE Transactions on Industrial Electronics*, vol. 53, pp. 1398–1409, Oct 2006.

[13] M. Malinowski, K. Gopakumar, J. Rodriguez, and M. A. Perez, "A survey on cascaded multilevel inverters," *IEEE Transactions on Industrial Electronics*, vol. 57, pp. 2197–2206, July 2010.

[14] J. I. Leon, S. Kouro, L. G. Franquelo, J. Rodriguez, and B. Wu, "The essential role and the continuous evolution of modulation techniques for voltage-source inverters in the past, present, and future power electronics," *IEEE Transactions on Industrial Electronics*, vol. 63, pp. 2688–2701, May 2016.

[15] H. S. Patel and R. G. Hoft, "Generalized techniques of harmonic elimination and voltage control in thyristor inverters: Part i–harmonic elimination," *IEEE Transactions on Industry Applications*, vol. IA-9, pp. 310–317, May 1973.

[16] H. S. Patel and R. G. Hoft, "Generalized techniques of harmonic elimination and voltage control in thyristor inverters: Part ii — voltage control techniques," *IEEE Transactions on Industry Applications*, vol. IA-10, pp. 666–673, Sept 1974.

[17] K. Yang, J. Hao, and Y. Wang, "Switching angles generation for selective harmonic elimination by using artificial neural networks and quasi-newton algorithm," in *2016 IEEE Energy Conversion Congress and Exposition (ECCE)*, pp. 1–5, Sept 2016.

[18] J. Sun and H. Grotstollen, "Solving nonlinear equations for selective harmonic eliminated PWM using predicted initial values," *Proceedings of the 1992 International Conference on Industrial Electronics, Control, Instrumentation, and Automation*, pp. 259–264 vol.1, Nov 1992.

[19] L. G. Franquelo, J. Napoles, R. C. P. Guisado, J. I. Leon, and M. A. Aguirre, "A flexible selective harmonic mitigation technique to meet grid codes in three-level pwm converters," *IEEE Transactions on Industrial Electronics*, vol. 54, pp. 3022–3029, Dec 2007.

[20] Javier Nápoles Luengo, "Flexible Pulse-Width Modulation Method for Multilevel Converters," May 2010.

[21] J. Napoles, J. I. Leon, R. Portillo, L. G. Franquelo, and M. A. Aguirre, "Selective harmonic mitigation technique for high-power converters," *IEEE Transactions on Industrial Electronics*, vol. 57, pp. 2315–2323, July 2010.

[22] AENOR, *UNE-EN 50160:2011: Voltage characteristics of electricity supplied by public electricity networks*. Madrid: AENOR, 2011.

[23] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, pp. 268–308, Sept. 2003.

[24] M. M. Hussain, H. Hattori, and N. Fujimoto, "A cuda implementation of the standard particle swarm optimization," in *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 219–226, Sept 2016.

[25] F. S. Abu-Mouti and M. E. El-Hawary, "Optimal distributed generation allocation and sizing in distribution systems via artificial bee colony algorithm," *IEEE Transactions on Power Delivery*, vol. 26, pp. 2090–2101, Oct 2011.

[26] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53–66, Apr 1997.

[27] W.-M. Lin, F.-S. Cheng, and M.-T. Tsay, "An improved tabu search for economic dispatch with multiple minima," *IEEE Transactions on Power Systems*, vol. 17, pp. 108–112, Feb 2002.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

[29] N. Ghorbani and E. Babaei, "Exchange market algorithm," *Applied Soft Computing*, vol. 19, pp. 177 – 187, 2014.

[30] NVIDIA, "CUDA 8 Performance Overwiev."

[31] MathWorks, "Academia -Mathwors and MATLAB Specifications."

[32] M. Ujaldón, "Programando la GPU con CUDA."

[33] N. Research, *CUDA Threads and Atomics*.

[34] NVIDIA Developer Zone, "CUDA Toolkit Documentation."

[35] I. C. E. Team, "Fast. Faster... Performance Comparison: C# (ILNumerics), FORTRAN, MATLAB and NumPy - Part II."

[36] I. Corporation, "Intel Core i7-4702MQ Product Specifications."

[37] NVIDIA, "NVIDIA GeForce GT 740M Product Specifications."

[38] I. Corporation, "Intel Core i7-4790K Product Specifications."

[39] NVIDIA, "NVIDIA GeForce GTX 780 Product Specifications."

[40] A. von Starck, A. Mühlbauer, and C. Kramer, *Handbook of Thermoprocessing Technologies: Fundamentals, Processes, Components, Safety*. Vulkan-Verlag, 2005.

[41] A. Ferreiro, J. Rodriguez, J. Salas, and Cendón, "CUSIMANN: An optimized simulated annealing software for GPUs,"

[42] N. Ghorbani, "Combined heat and power economic dispatch using exchange market algorithm," *International Journal of Electrical Power & Energy Systems*, vol. 82, pp. 58 – 66, 2016.

[43] G. Caldarelli, M. Marsili, and Y. C. Zhang, "A prototype model of stock exchange," papers, arXiv.org, 1997.

[44] N. Ghorbani and E. Babaei, "Exchange market algorithm for economic load dispatch," *International Journal of Electrical Power & Energy Systems*, vol. 75, pp. 19 – 27, 2016.

[45] N. Ploskas and N. Samaras, *GPU Programming in MATLAB*. Boston: Morgan Kaufmann, 2016.

[46] "IEEE Recommended Practice on Surge Testing for Equipment Connected to Low-Voltage (1000 V and Less) AC Power Circuits," *IEEE Std C62.45-2002 (Revision of IEEE Std C62.45-1992)*, 2003.

[47] dSPACE GmbH, "dSPACE - Industries and Solutions."

[48] dSPACE GmbH, "DS1007 PPC Processor Board."

[49] B. C. Becker, "Analysis of JPEG Decoding Speeds."